FAMU-FSU College of Engineering
Department of Electrical and Computer Engineering

EEL4911C – ECE Senior Design Project I

FINAL REPORT

Project Title: Turf-Tec Metrology

Team # 4

Student Team Members:

Shaneetra Graham, Computer Engineering (Email: sjg10d@my.fsu.edu)
Christian Rodriguez, Computer Engineering (Email: cgr09@my.fsu.edu)
Joyce Kosivi, Electrical Engineering (Email: Joycekosivi@yahoo.com)
Johnnie McCormick, Electrical Engineering (Email: jrm08m@my.fsu.edu)
John Rodriguez, Electrical Engineering (Email: jr11k@my.fsu.edu)

Senior Design Instructor
Dr. Michael Frank

ECE Review Committee Members
Petru Andrei
Rajendra Arora

External Client
John Mascaro

Continuity
Jonathan Casanas

Submitted in partial fulfillment of the requirements for
EEL4911C – ECE Senior Design Project I

November 14, 2013

Executive Summary

The project's objective is to provide a working Impact Tester that can be mass-produced and highly reliable. The impact tester is a measurement tool used to measure how hard a surface is. This impact tester will be more specifically marketed towards use on turf and sports fields. Turf Tec international is the leading sales representative for the Impact Tester and will direct their sales to the market of their choice. The main market targeted by the company is the National Football League (NFL). The design focus of this project will be centered on the Clegg Impact Tester, which is currently being sold as the primary Impact Tester by Turf Tec. The Impact Tester will be tested using the standard methods published by the American Society for Testing and Materials (ASTM). The method F355 requires a weight to be dropped from a standard height and the accelerometer measures the impact. The procedure F936 is under the same method and requires three successive drops on each test point. The GMAX value is the average of the second and third drop. This testing strategy will test the mechanical and electrical capabilities of the Impact Tester. For the mechanical portion, the missile will need to weigh five pounds and be lifted from eighteen inches. These dimensions are scaled to properly demonstrate a human head of twenty pounds dropping from a height of two feet. The electrical portion of the Impact Tester provides three main functions. For one, it reads the GMAX value which is calculated as a result of impact with the missile and the ground. Second, it logs the location of impact so multiple tests can be conducted at a time. Lastly, it displays the data in a user-friendly display. These main functions are broken down into a handful of electrical components, which are the primary focus of the Turf Tec senior design team. An accelerometer converts the impact into an electrical pulse, which is then converted to appropriate GMAX values. The GPS system calculates location, date and time and the user interface is in the form of an LCD screen with pushbuttons and LEDs to send and receive information, respectively. Certain elements of the Impact Tester will differ from the Clegg Impact tester. For one, the GPS feature is a requested addition. The ability to transfer data through a USB cord rather than removing the SD card inside is also an upgrade requested. Simplified menu features, water resistance, and rechargeable batteries are additions as well. The elements of the final prototype will be transferred to a printed circuit board design using a Computer Aided Design (CAD) tool.

As stated, this PCB will incorporate all electrical functions of the Impact Tester onto one board or possibly two, if the power supply is made a separate board. The reason for this merge of elements is to ensure that the Impact Tester designed will be difficult to duplicate. The PCB layout will be designed by the Turf Tec senior design team and then sent for production. Along with providing a working schematic at the end of manufacturing, the ability to mass-produce the PCB board is also a responsibility of the team. Elements chosen should be cost-effective to ensure that the final product could sell for a decent price.

The product will also promise durability both electrically and mechanically. Turf Tec International will do the mechanical production. The electrical components will be placed in a box, which will then be attached to the missile. This box will require water resistance and toughness due to the market that Mr. Mascaro is directing his product to. Most likely, the Impact Tester will be thrown around in the back of trucks, or left outdoors or in sheds. Water, strength, and excessive heat are factors that are taken into account when making decisions

The impact tester is a measurement tool used to measure how hard a surface is. This impact tester will be more specifically marketed toward use on turf and sports fields

Contents

Executive Summary 2

1 Introduction 7

1.1 Acknowledgments 7

1.2 Problem Statement 7

1.3 Operating Environment 7

1.4 Intended Use(s) and Intended User(s) 7

1.5 Assumptions and Limitations 8

1.5.1 Assumptions 8

1.5.2 Limitations 8

1.6 End Product and Other Deliverables 8

2 System Design 9

2.1 Overview of the System 9

2.2 Major Components of the System 10

2.2.1 Arduino Mega2560 10

2.2.2 Accelerometer 11

2.2.3 Battery (Power System) 12

2.2.4 Global Positioning System (GPS) 13

2.2.5 Data Logger/ Micro SD 14

2.2.6 User Interface 15

2.2.7 Printed Circuit Board 16

2.3 Subsystem Requirements 18

2.3.1 Requirements Specification for Arduino Board 18

2.3.2 Requirements Specification for GPS 19

2.3.3 Requirements Specification for Accelerometer 19

2.3.4 Requirements Specification for Data Logger 19

2.3.5 Requirements Specification for User Interface 19

2.3.6 Requirements Specification for Power System 19

2.3.7 Requirements Specification for Printed Circuit Board 20

2.4 Performance Assessment 21

2.5 Design Process 22

3. Design of Major Components/Subsystems 22

3.1 Arduino MEGA 23

3.1.1 Status of Microcontroller 23
3.2 GPS 30
3.2.1 Status of GPS 30
3.2.2 Status of Pin Selections 32
3.2.4 Status of backup battery 33
3.2.5 Status of Connecting to the Arduino Mega2560 34
3.2.6 Status of Packet Implementation 39
3.2.7 Overall Status 40
3.3 Accelerometer 40
3.3.1 Status of Accelerometer 40
3.3.2 Status of Software Interface 40
3.3.3 Status of ICP (Integrated Circuit Piezoelectric) 46
3.3.4 Status of Alternatives Considered 47
3.3.5 Overall Status 48
3.4 Data Logger 48
3.4.1 Status of Data Logger 48
3.4.2 Status of Connection to the Ardunio Mega 48
3.4.3 Status of Communicating with the USB Port 54
3.4.4 Status of PC Software 57
3.4.5 Status of Alternatives Considered 58
3.4.6 Overall Status 58
3.5 User Interface 59
3.5.1 Status of User Interface 59
3.5.2 Status of Liquid Crystal Display 59
3.5.3 Status of Light-Emitting Diodes (LEDs) 63
3.5.4 Status of Push Buttons 66
3.5.5 Status of Alternative Consideration 71
3.5.6 Overall Status 71
3.6 Power System (Battery) 71
3.6.1 Status of Power System 72
3.6.2 Status of Battery 75
3.6.3 Status of Alternatives Considered 76
3.7 Printed Circuit Board 76

3.7.1 Status of Printed Circuit Board 76
3.7.2 Status of Software Selected 76
3.7.3 Status of Interface Layers 77
3.7.4 Status of Unnecessary Components 79
3.7.5 Status of Alternatives Considered 79
3.7.6 Overall Status 80
4 Schedule 80
5 Budget Estimate 81
6 Overall Risk Assessment 82
6.1 Technical Risks 82
6.1.1 Technical Risk: Arduino MEGA2560 82
6.1.2 Technical Risk: GPS 83
6.1.3 Technical Risk: Accelerometer 85
6.1.3 Technical Risk: Accelerometer 86
6.1.4 Technical Risk: Data Logger 87
6.1.5 Technical Risk: User Interface 87
6.1.5.1 Technical Risk 1: LCD does not display correct information 87
6.1.6 Technical Risk: Power System 89
6.7.1 Technical Risk: Printed Circuit Board 91
6.7.1.3 Technical Risk 3: Improper Design 92
6.2 Schedule Risks 93
6.2.1 Schedule Risk: Behind Schedule 93
6.2.2 Schedule Risk 2: Need to re-order a damaged component 94
6.2.3 Schedule Risk 3: Prototype isn’t finished prior to winter break 94
6.2.4 Schedule Risk 4: The transition from one component to another will not be smooth. 95
6.3 Budget Risk 95
6.3.1 Budget Risk 1: More components must be purchased 96
6.3.2 Budget Risk 2: The price for a component was underestimated 96
6.3.3 Budget Risk 3: Previous component needs to be re-ordered 96
7 Conclusion 98
8 References 98

1 Introduction

1.1 Acknowledgments

This year's Turf-Tec International senior design team would like to acknowledge the previous senior design teams for their contributions and time given to this prototype. The team would also like to acknowledge Jonathan Casanas for all his work he has continued to do on this prototype and also for assisting this year's team with technical aspects of the prototype. The team would also like to acknowledge Dr. Frank for all his contributions to the project and also helping this year's team understand the coding of the Arduino. The team would like to give a special acknowledgement to Mr. Mascaro for providing the idea of the impact tester as well as donating $1000.00 this semester for to the team to complete the product.

1.2 Problem Statement

The objective for the Turf-Tec senior design team is to update and fix minor bugs in the existing prototype. The team will also be rearranging the design of the circuit board as to create a less error-prone, easily maintained, and organized design as Mr. Mascaro stated that he would like the product to be mass produced and easily assembled for production. A few of the problematic items the team will have to fix would include the bug last year's team introduced into the firmware software and also the placement of the coaxial cable.

1.3 Operating Environment

The impact tester will be exposed to many different environments. Although it will mainly be used for outdoor purposes, the impact tester will also be used for indoor environments such as NFL stadiums that are enclosed like domes or just indoor practice facilities. While outdoors, the impact tester can be exposed to many different elements such as extreme heat, direct sunlight, rain, and cold. Considering its end users, groundskeepers and maintenance crews most likely, the product may likely be thrown, dropped and overall treated roughly. For storage, the impact tester may be stored in a cool storage unit or in an area where the temperature could vary widely depending on the outside climate. With a wide range of operating environments, the impact tester must be ready for all possibilities.

1.4 Intended Use(s) and Intended User(s)

The Impact tester will be sold by Turf-Tec International to a wide range of Mr. Mascaro's clients who work in the sports industry such as the NFL, NCAA, and golf courses. With the alarmingly increasing number of diagnosed head injuries stemming from contact sports many of these sports industries are scrambling to understand and hopefully prevent them. The NFL for example, has a set of regulations on how hard a surface of the field can be which an upper limit value of 200 GMAX. If the surface exceeds 200 on any part of the turf's surface is should be replaced.

The intended use of the impact tester is to the test the hardness of a turf surface. Operating the impact tester will be very simple. The user should understand that the tester will only be accurate for the surface that is being tested and not any area beyond that unless that area can be proven to be of the exact same material as the surface tested. The tester needs to be on a level surface in order to be most accurate, and should be placed upright and be able to stand still by itself. The user will turn on the device using the power button on the controlling component and the LCD should display the date and time. The GPS will obtain the devices current coordinates and the date and time. At this point, the user will raise the missile, a weight with an accelerometer attached at its base, to an indicated level and release it. The missile should drop down through the guide tube and the LCD will display the GMAX measurement once the missile lands.

1.5 Assumptions and Limitations

1.5.1 Assumptions

1. Battery will last for several hours
2. Data logger has adequate space for storing multiple readings
3. LCD screen must be able to be viewed in direct sunlight
4. GPS will measure latitude and longitude coordinates
5. GPS will have an error no higher than 10 feet in any direction
6. Impact tester should be water resistant
7. USB will transfer data collected to PC computer
8. Accelerometer will measure the GMAX values
9. Arduino powered by battery source

1.5.2 Limitations

1. Not exceeding manufacturing cost of $300.00
2. Impact tester durability to withstand being dropped
3. Precise locations of measurements
4. Adding Bluetooth to impact tester

1.6 End Product and Other Deliverables

The Turf-Tec senior design team will complete the existing prototype. The prototype will be delivered at the end of the spring semester 2014. Turf-Tec International will be able to mass manufacture the new prototype as demand fills for this product. Other deliverables include a power supply for the battery, a user manual on how to properly use the product, another manual for Turf-Tec International for setup purposes, as well as any other documentation on the product.

2 System Design

2.1 Overview of the System

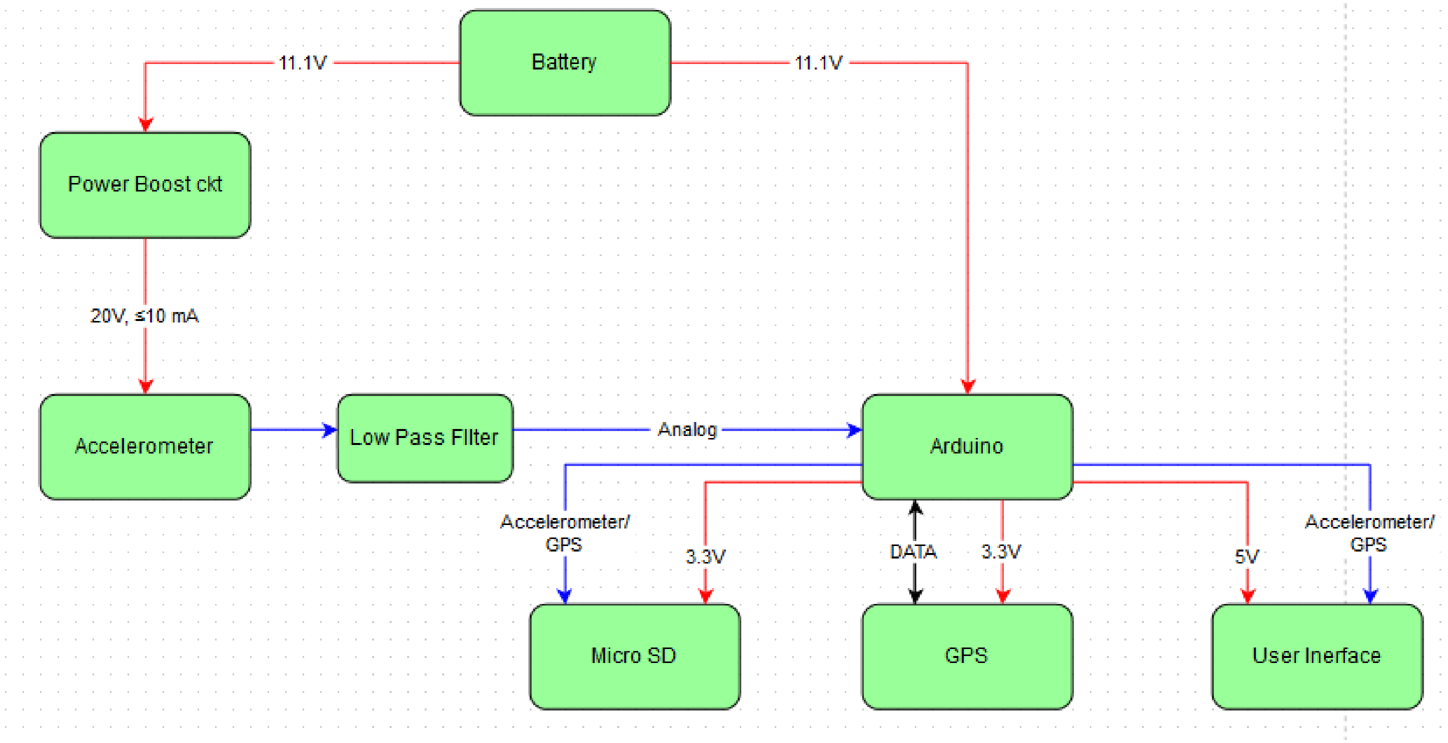


Figure 1: Top Level Diagram

The Impact Tester will consist of six main components that work together to provide the correct functions. The battery provides power to all the components. The heart of the design is the Arduino. The Arduino is the component that will be communicating with the GPS, data logger, accelerometer, and the user interface. It instructs the GPS to retrieve the correct coordinates and then store them in the data logger. The Arduino, similarly, communicates to the accelerometer to give its readings and convert them into GMAX values then store them into the data logger. Information stored in the data logger can be transferred to an external CPU. The Arduino will work directly with the user interface to display the measurements on the LCD.

2.2 Major Components of the System

2.2.1 Arduino Mega2560

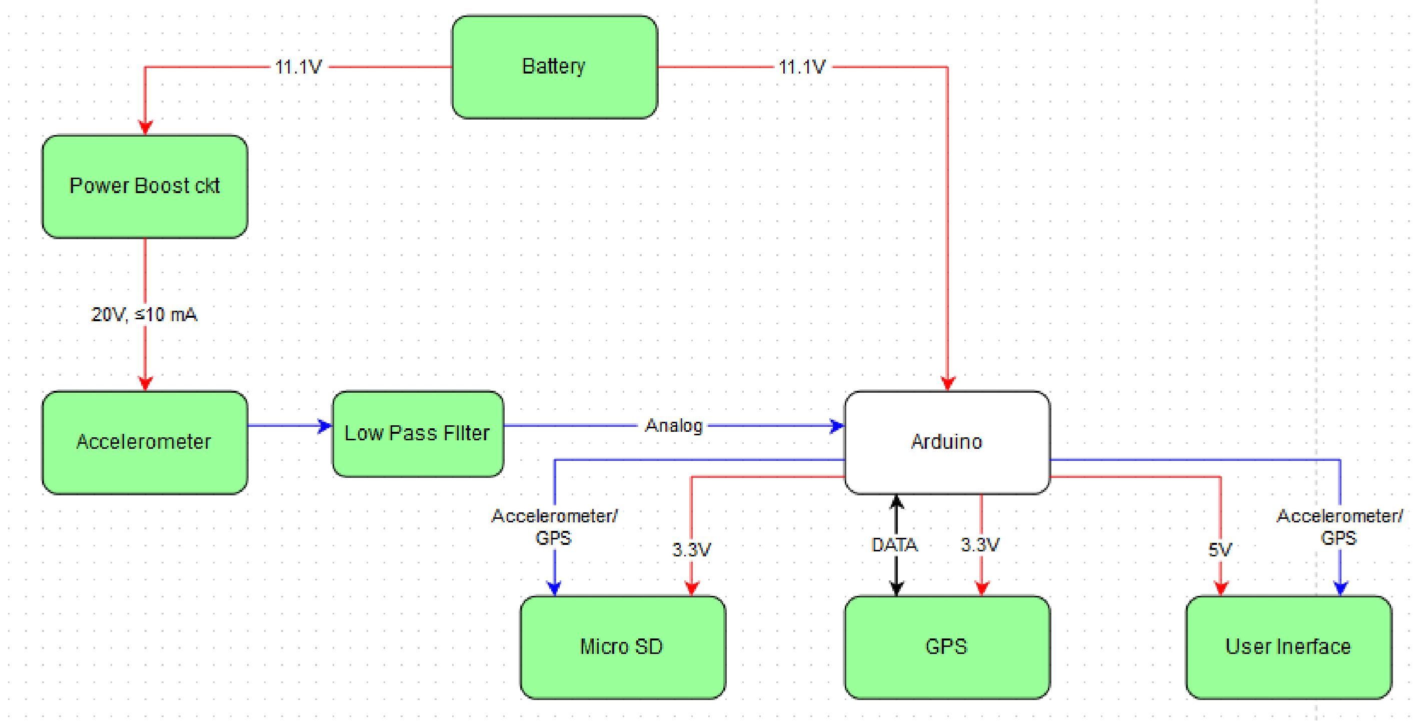


Figure 2: Microcontroller (Arduino Mega2560)

The microcontroller is the main component of the design. It controls all the functions of the other components and directs the flow of information in the design. The data logger, accelerometer, and GPS act as inputs into the microcontroller and the user interface acts as an output of the controller. Arduino is an open-source physical computing platform based on simple I/O board and a development environment that implements the Processing/Wiring language. The Arduino MEGA2560 features 54 digital I/O pins (14 PWM outputs), 16 Analog Inputs, 256K flash memory with a 16 MHz clock speed. Due to the effectiveness of this board, this year’s team found no reason to need to look for another board and will be using it in the final prototype.

2.2.2 Accelerometer

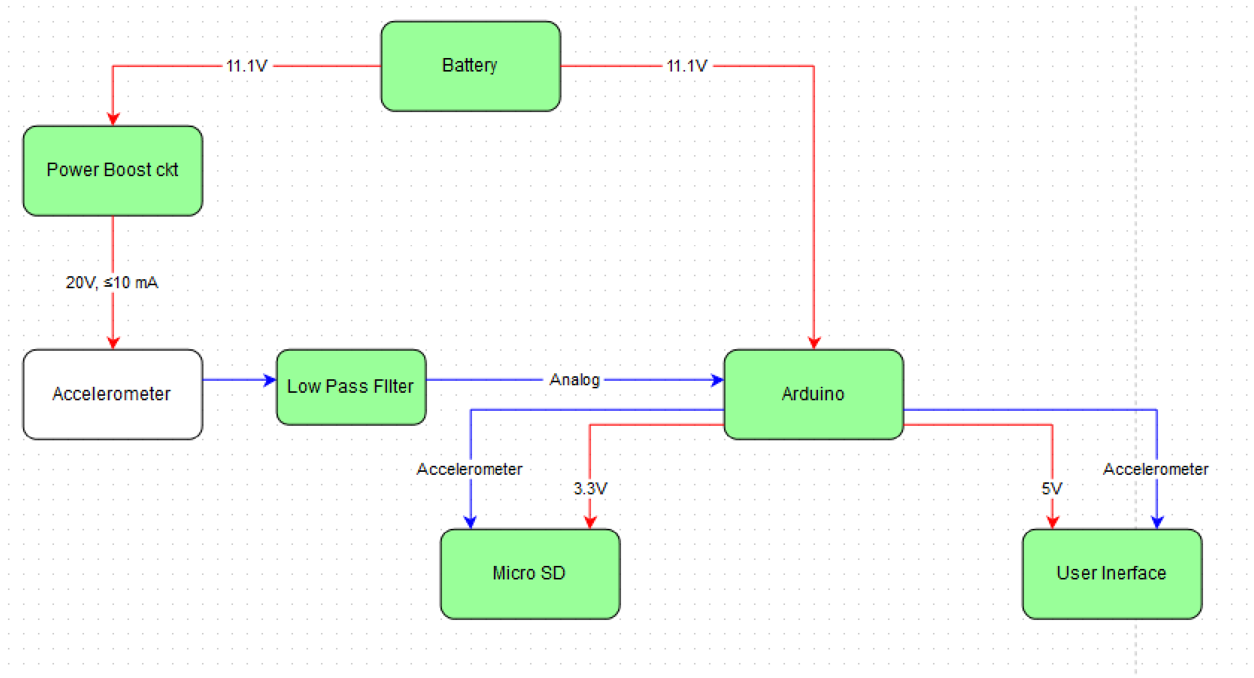


Figure 3: Overview of Accelerometer

The accelerometer might be the single most important component of the overall design. This component provides a raw, instantaneous acceleration measurement which the CPU then has to calculate the GMAX readings from the raw signal. The accelerometer can obtain a small amount of spectral noise meaning more accuracy. An ICP (Integrated Circuit Piezoelectric) that is needed for the impact tester will have to have low impedance charge output. When used, the Arduino board function should then convert the voltage by 10mV/g. The accelerometer plays a large role in the measurements of the surfaces; and with the accelerometer being accurate it can prevent any fatal trauma to the head, depending on the field being analyzed.

2.2.3 Battery (Power System)

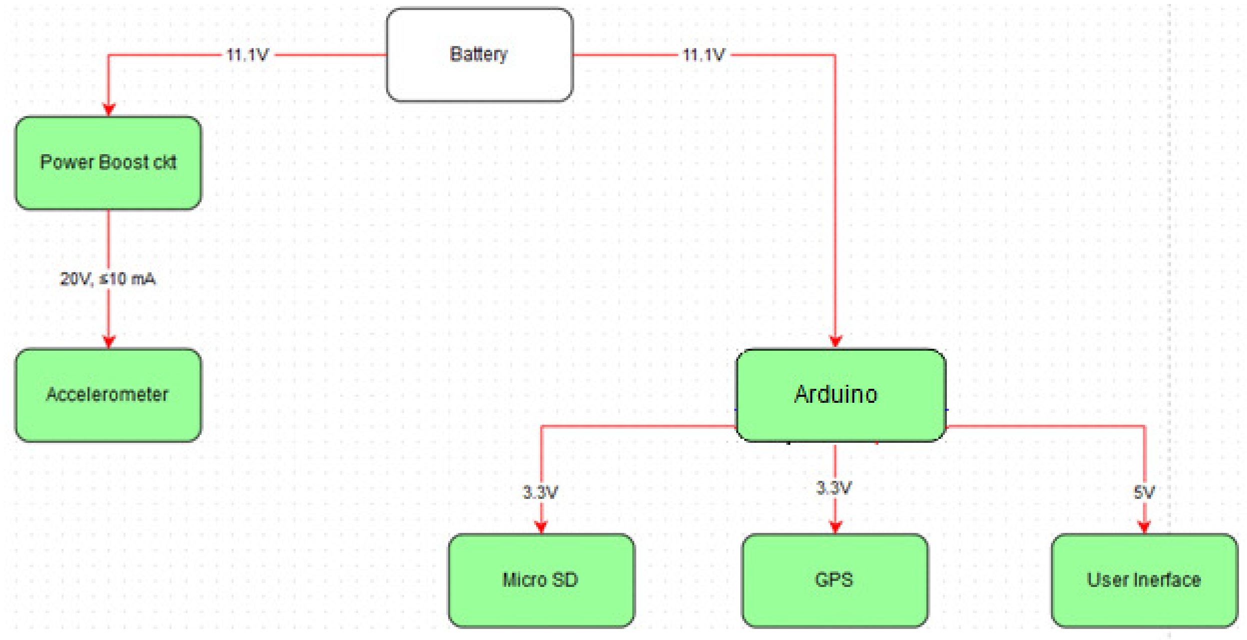


Figure 4: Battery diagram

The power system is the circuit that powers the entire system. It provides the Arduino Mega 2650 and the accelerometer with the necessary amount of power for each component to function. The power system will consist of a rechargeable lithium ion battery to provide the power. The Tenergy Li-Ion 11.1V battery is made of three 2600mAh cylindrical 18650 cells with PCB and poly switch for full protection. There is a built-in IC chip that will prevent the battery pack from over charging (protection at 13V) and over discharging (protection at 7.2V) and prolong battery life. This year's team decided to make its on boost regulator to prevent having problems with too high or too low voltage.

2.2.4 Global Positioning System (GPS)

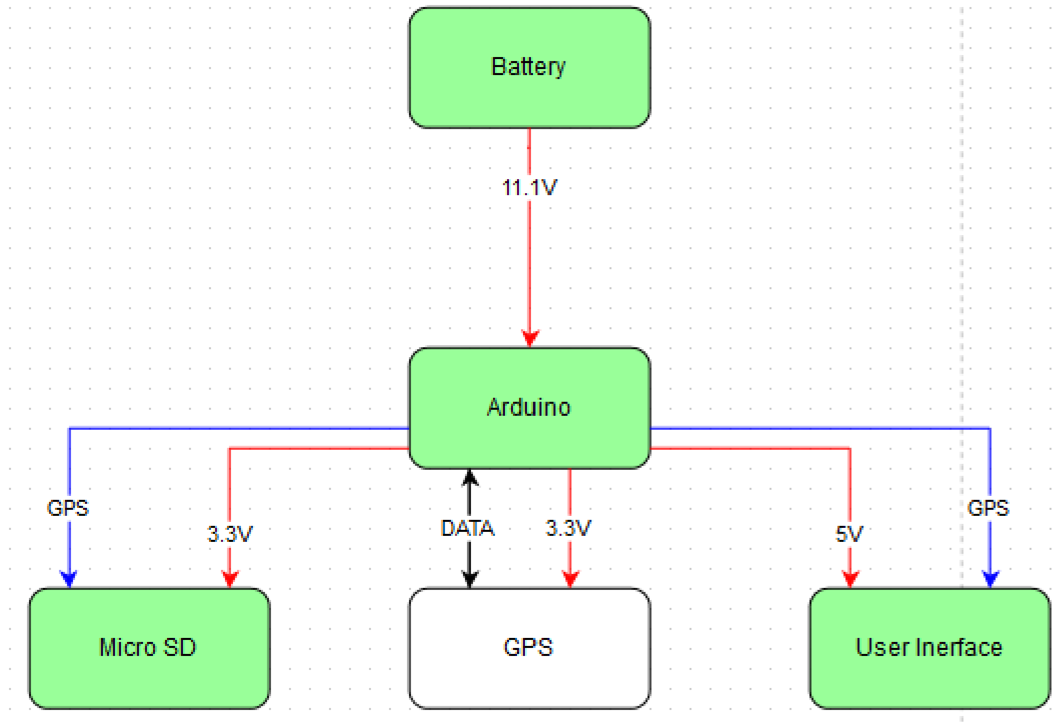


Figure 5: GPS Diagram

The GPS is used to record location, date, and time of each testing experiment. This is an important aspect of the Senior Design Impact Tester, for the Clegg Impact Tester currently being sold by Mr. Mascaro does not include a GPS. This added feature is important for conducting multiple tests in different locations, for example: different yard lines on a football field or different greens on a golf course. The LS20031 GPS features MediaTek high sensitivity solutions, supports 66-channel GPS, had an update rate up to 10Hz, it is capable of SBAS (WAAS, EGNOS, MSAS) also known as differential GPS, it has a built-in micro battery to reserve system data for rapid satellite acquisition, and it has an LED indicator for GPS fix or not fix.

2.2.5 Data Logger/ Micro SD

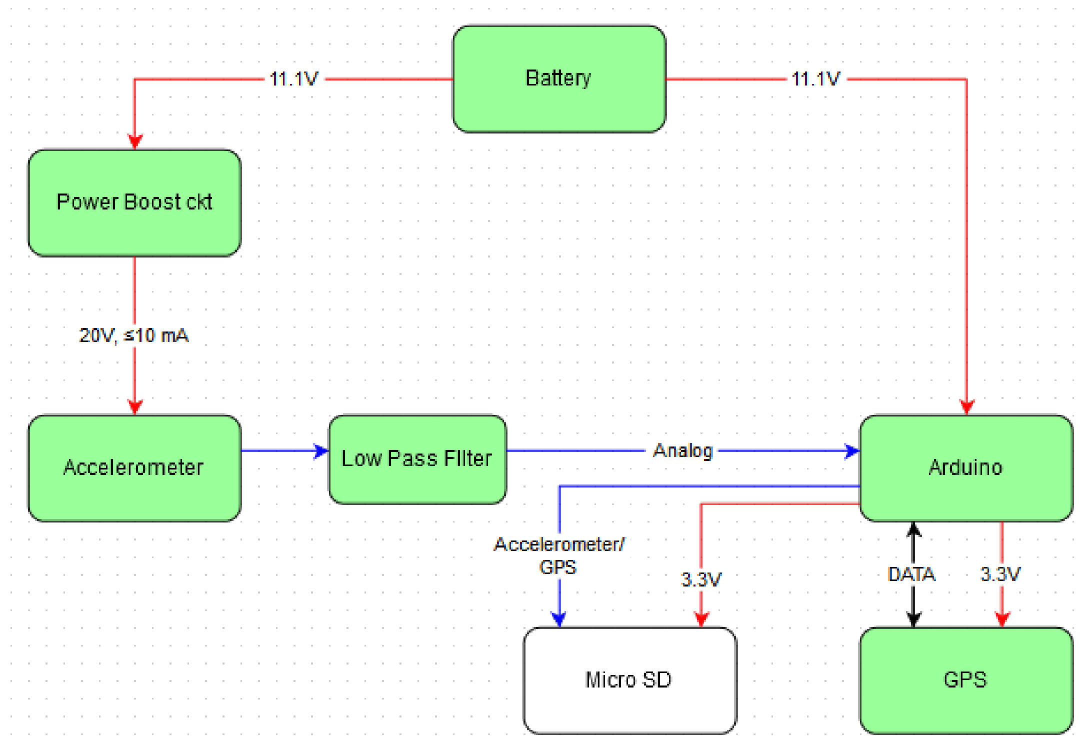


Figure 6: Micro SD/ (data logger)

The data logger functionality is provided with the help of an external memory component use to store information or readings that are obtained from the accelerometer and Global Positioning System (GPS). The micro SD shield operates using 3.3V input voltage and the power draw when writing to the card can be up to 100mA. The micro-SD shield will connect directly to pins 3.3V, GND, 50, 51, 52 and 53 located on the Arduino Mega board. The memory being stored is coordinates, date and time from the GPS and the converted GMAX readings from the accelerometer. This data will then be saved as a CSV file on the Micro SD card and then extracted through the serial port from the Arduino.

2.2.6 User Interface

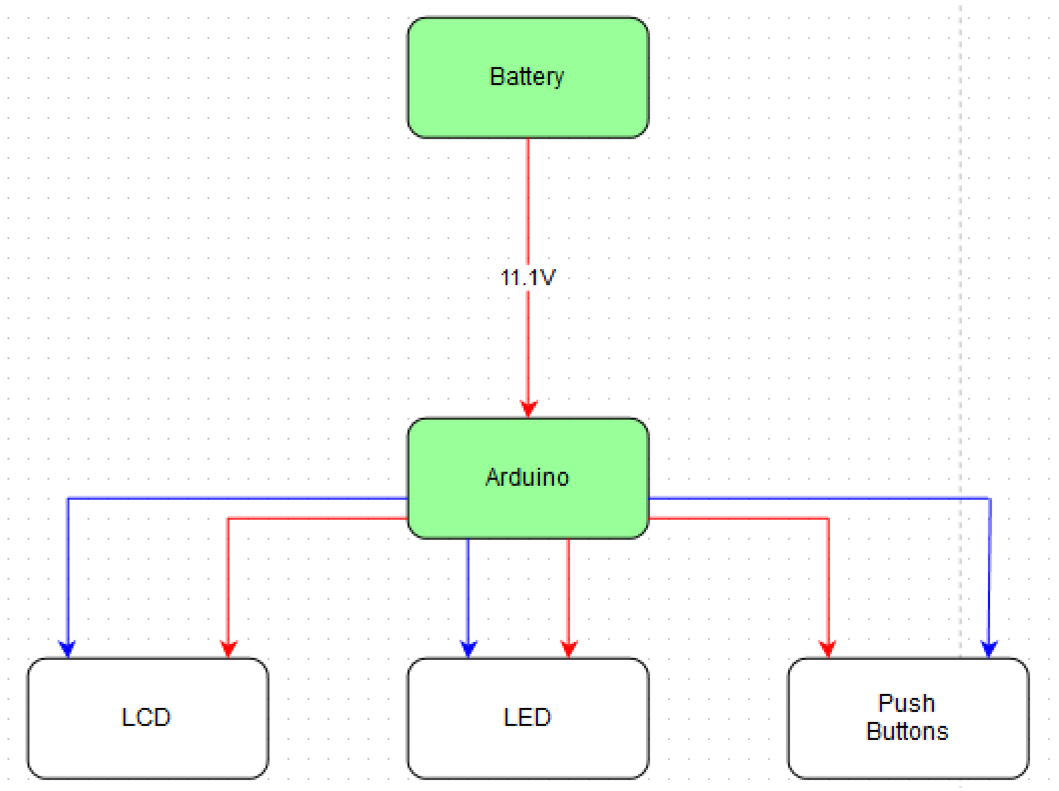


Figure 7: User interface Diagram

The user interface is the main interaction between the user and the Impact Tester. It consists of a liquid crystal display, four push buttons, and two light-emitting diodes. It is used to display the GPS coordinates, the GMAX values, the date and time, and the ready/hold state. This allows the user to see exactly what is being observed during each Impact Tester run. The figure below shows the main screen layout.

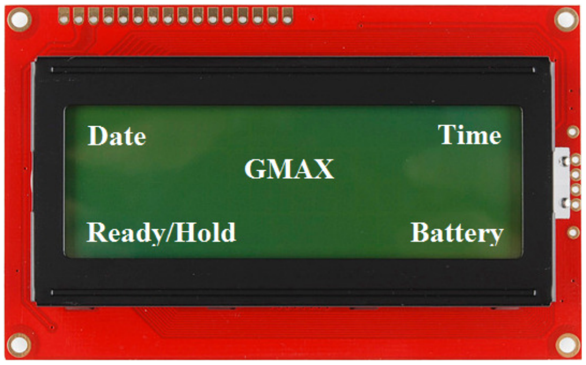


Figure 8: main screen layout of LCD

Each push button has its own independent functions. The purpose of the largest push button is to toggle the system between the on and off state. Secondary push buttons will save and clear the data during each run respectively, allowing the user to have control of the measurements needed. When pressing these two buttons at the same time, the LCD will enter a mode which the user can adjust the time zone accordingly. This feature runs on a simple delay so after the time zone has been set the user will wait for 5 seconds and the LCD will return to the normal menu screen. The fourth pushbutton is a toggle to switch between two menu screens on the LCD. The first screen will show the main menu which has everything listed above except for the GPS coordinates, and the second screen will display the current GPS coordinates at the center of the screen. When the button is held, the screen will take the form of the layout below until released

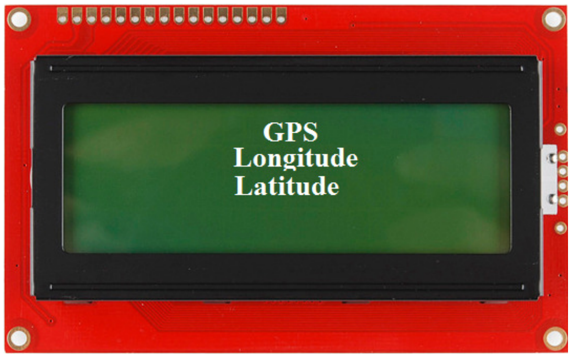


Figure 9: shows the way the GPS data will be displayed

Finally, light-emitting diodes (LEDs) are used to indicate information and the state of the process. One LED (green) will indicate when the data is being acquired and the other LED (red) will indicate when the data is being saved or if any errors have occurred during setup.

2.2.7 Printed Circuit Board

Copper thickness: The thickness and width of a trace determines the amount of current in amps the trace can safely carry. The thickness is also used in the calculation of trace impedance in ohms. The formula for copper weight in ounces to thickness in mils conversion:

Thickness in Oz = thickness in mils /1.37'

The following is the formula for thickness in mils to cooper weight in ounces conversion

t in mils = t in Oz * 1.37

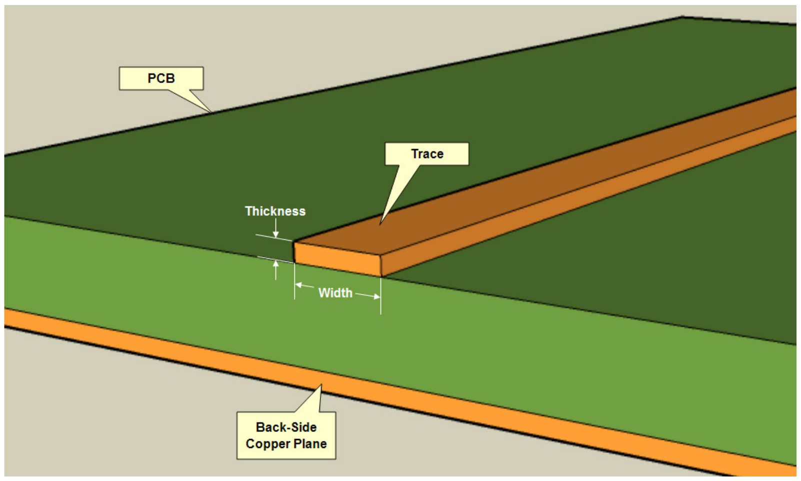
Weight	Thickness
½ Oz	0.7 mils
1 Oz	1.4 mils
2 Oz	2.8 mils

Footprints: A way of drawing with all the components with the manufacturer information with all the details of the position of each element that would be soldered onto the board.

Plated-Hole size tolerance of +/- 0.0005”. The board has a specific dimension tolerance of +/- 0.010”. Plating thickness in the hole wall of 0.0008” minimum allowed. Many of the components in the design are through-hole requiring a hole for each pin.

Circuit Board: Overall thickness of most PCBs are 1/16 inch, but can go as small as 1/32 inch. Printed Circuit Board can consist of many layers and layers on top of other layers.

Trace clearance: The spacing between all the traces is super important factor in the printed circuit board design. The traces on the board can short out if the traces are too close.



$$Area = \frac{\text{Current in Amps}}{(((\text{K} * \text{temperature}^{\text{c}}\ )^{b})\text{\^{}}(^{1}/c))}$$

$$Width\ = \frac{\text{Area\^{}2}}{thickness * 1.378}$$

Trace clearance, the distance between printed circuit boards traces is critical to avoid a disruptive electrical discharge around or over the surface of a solid insulator. The industry safety standards prescribe different spacing depending on the Voltage, Current, Heat and other factors should be considered.

Gerber files are a standard electronics industry file format used to communicate design information to manufacturing for many types of printed circuit boards. In many ways Gerber is the electronics world's equivalent of PDF.

2.3 Subsystem Requirements

2.3.1 Requirements Specification for Arduino Board

1. The Arduino MEGA2560 is responsible for:
    a. Receiving information from the GPS and converting it to display date/time and coordinates on the LCD screen.
    b. Receiving the electrical impulse from the accelerometer and converting it to GMAX values to be displayed on the LCD screen.
    c. Communicate with the LEDs to flash when receiving information from the GPS and when saving information onto the data logger.
    d. Communicate with the pushbuttons. One will power all the other subsystems, another will save data received, another will clear it and the last one will toggle screens.
    e. Use the USB port to pull data saved onto the data logger and display it in a compatible format on a computer.
    f. Receive voltage from the battery and divide the power within the board, specifically 5V to the user interface, 3.3V to the GPS, and 1.8 to the ATmega2560 microcontroller. * Note: In the final subsystem, this voltage division will be part of the power system in general. The same concept that is used for the MEGA will be done for the power system, however, the Arduino MEGA 2560 will not exist following integration and fabrication of the printed circuit board.
2. The ATmega2560 microcontroller requires 500uA @ 1.8V when in active mode and .1uA @ 1.8V when on standby

3. The 256K flash memory requires 8kB of it to be for the boot loader
4. The USB input supplies up to 500mA. Any more than that can result in serious damage.
5. All pins in the Arduino are defaulted as input pins. As an input pin, the current is very close to zero. However, when a pin is declared as an output it decides how much current to draw in response to an applied voltage.
    a. Each pin has a max current draw of 40mA
    b. The 3.3V pin has a max current draw of 50mA
    c. The 5V pin has a max current draw of 40mA
6. The MCU is responsible for receiving data from different components and converting it for better readability. For example:
    a. Receiving information from the GPS and converting it to display date/time and coordinates on the LCD screen.
    b. Receiving the electrical impulse from the accelerometer and converting it to GMAX values to be displayed on the LCD screen.
    c. Communicate with the Arduino MEGA2560 to transfer information received from the GPS and Accelerometer to the user's computer utilizing a USB port.
    d. Use the USB port to pull data saved onto the data logger and display it in a compatible format on a computer.

2.3.2 Requirements Specification for GPS

1. The LS20031 GPS is responsible for:
    a. Receiving satellite information- specifically date, time and longitude and latitude coordinates.
2. The LS20031 GPS requires 41mA @ 3.3V to function.
3. Uses NMEA 0183 version 3.01 as its protocol support. Default bit rate is 9600 bps, receiving 8 data bits. NMEA output messages include global positioning system fixed date (GGA), geographic position - latitude/longitude (GLL), GNSS DOP and active satellites (GSA), GNSS satellites in view (GSV), recommended minimum specific GNSS data (RMC), and course over ground and ground speed (VTG).
    a. The only ones that will be used are GGA, for the coordinates, and RMC, for the date.

4. The built-in micro battery reserves system data for rapid satellite acquisition.

2.3.3 Requirements Specification for Accelerometer

1. Accelerometer (Model 353C04) Responsibilities:
    a. Instantaneous raw acceleration G-values
    b. Output signal must have a voltage signal ranging from 0-5V
    c. Has be powered between 18 – 30V and have a current from 2-20mA

2.3.4 Requirements Specification for Data Logger

a. Micro SD shield responsible for storing data
b. Micro SD requires a minimum input voltage of 5V and a max current of 100 mA

2.3.5 Requirements Specification for User Interface

1. The LCD-00256 is used for:
    a. Displaying the date, time, and coordinates in correspondence to the GPS
    b. Displaying the correct GMAX values in correspondence to the Accelerometer
    c. Displaying the battery level of the power supply as obtained from the Arduino
2. The LCD-00256 requires:
    a. A minimum of 4.7V and a maximum of 5.5V
    b. A maximum current of 3mA
    c. Operating temperature range from 0 to +50 degrees Celsius
3. The Diffused RGB (tri-color) LEDs are responsible for indicating when the information is being gathered and saved.
4. The Diffused RGB (tri-color) LEDs require:
    a. Red: 2.1-2.5V Forward Voltage, at 20mA current
    b. Green: 3.8-4.5V, at 20mA current
    c. Blue: 3.8-4.5V, at 20mA current

2.3.6 Requirements Specification for Power System

1. The power system is responsible for:

a. Provide the Arduino Board with 7 – 12V necessary to function.
b. Provide the accelerometer with the 10 – 15mA and 25.9V needed to function
c. Battery must last for several hours

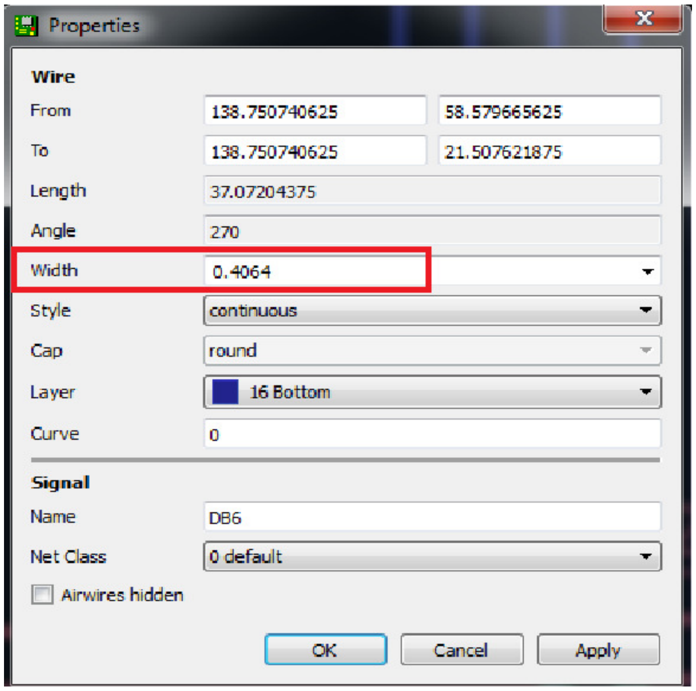
2.3.7 Requirements Specification for Printed Circuit Board

a) Cooper thickness: This is total thickness of copper on the board surface. Copper weight is measured in oz /sq. foot. 1 Oz = a minimum of 0.0012” thickness.
b) Footprints: The footprints are the list of components with all the specifications like item number, voltage and the name of the manufacture.

c) Holes: In the printed circuit board there are three different types of holes, the through hole, layers hole, drill holes.

Through hole size 0.015 mil
Layers hole size 0.06 mil
Drill hole size 3.2 mm

d) Board Thickness and Layers: The standard base thickness is 1/16 inch 0.062"
e) Trace width: The interface design on eagle cad has a trace width of 0.4064



Wire

From	138.750740625	58.579665625
To	138.750740625	21.507621875
Length	37.07204375	
Angle	270	
Width	0.4064	
Style	continuous	
Cap	round	
Layer	16 Bottom	
Curve	0	

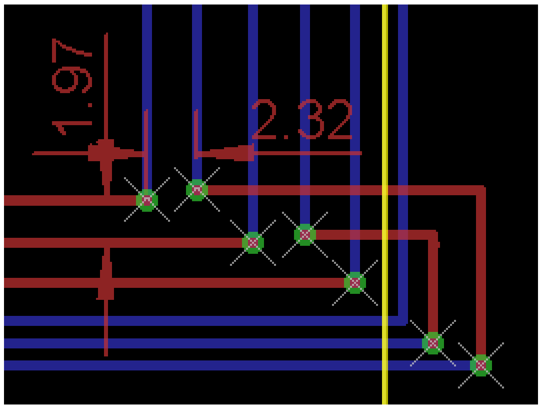
Signal

Name	DB6
Net Class	0 default

☐ Airwires hidden

f) Trace clearance: To prevent shorting, ground and power layer clearances need to be a minimum over the finished hole size of 0.020” to 0.025”.
The clearance between the traces on the top level is 2.32mm.

The clearance between the traces on bottom level is 1.97mm.



g) Gerber files: There are two standards Gerber files the old version called RS-274D and the newest version called RS-274X. The new version comes with more features that would embed aperture information into the files.
List of the new features:

- Gerber files runs on Linux.
- Support complex layers operations.
- Different measurement units supported inch, mm and mil.
- Multiple files can be loaded.
- Set the color layers independently using pop color box.

Gerber Data - Layer PCB –File Name	**Description**
PCB-Interface. TOP	Top cooper layer
PCB-Interface.BOT	Bottom cooper layer
PCB-Interface.SMT	Solder mask Top
PCB-Interface.SMB	Solder mask Bottom
PCB-Interface.SST	Silk Screen TOP
PCB-Interface.DRD	Drill Drawing
PCB-Interface.TAP	Excellon Drill File
PCB-Interface.IIS	Apeture List
PCb-Interface.TXT	This read me file

Fig 3.2.7 All the Gerber files of turf-Tec design.

2.4 Performance Assessment

The impact tester was made with regards to the needs analysis and requirements specifications document. All of these major electrical components must be able to perform certain task in order for the Impact tester to function properly. The table below shows the needs of what each component is required to do.

#	Electrical Requirement	MEGA 2560	GPS	Accel.	User Interface	PCB	Power Supply	Micro SD Card
1	Device needs to be able to read in data from the accelerometer and interpret the data.	√		√	√			
2	Device needs a proper user interface in order to ensure “friendliness” of the product.	√			√			
3	Device needs to read coordinates recorded from the GPS device.	√	√					
4	Device needs to have an internal source of lasting power (preferably batteries) with the ability to display correct battery amount.						√	
5	A printed circuit board design must be designed to include all required hardware needed to perform the functions of the Impact Tester.					√		
6	Device needs to store information in the data logger and be able to retrieve that information through the serial port.	√						√

2.5 Design Process

The most important part of this year’s Turf-Tec team was upgrading the PCB. The prototype from the previous year was a working prototype at one point. We decided, with the help of Dr. Frank and Jonathon that condensing the circuitry would be better. This was our biggest challenge, to have a prototype that worked correctly and was manufacturable.

We decided to take advantage of the user interface board. Our goal is to have everything connected on that single board. Front side consisting of the LCD, the pushbuttons, and the power boost circuit. The back of the user interface is where will plug in the Arduino. The GPS will plug directly into an open space on the back of the user interface board where its connections will be wired to the Arduino. The same idea will be used for connecting the Micro SD.

This reason behind this design is to make for a simple assembly.

3. Design of Major Components/Subsystems

The Turf-Tec team is responsible for all the electrical work that goes into the new impact tester. All electrical components are considered major for this group, and with that each member is responsible for one or two of them. The components are split up as follows:

3.1 Arduino MEGA

**All technical risks can be seen in the Risk Assessment section of this report.*

3.1.1 Status of Microcontroller

Because this is an ongoing project there was an already working microcontroller when this year's team began working the project. The current Microcontroller being used is the Arduino MEGA2560, same as last year's. This board features 54 digital I/O pins (14 PWM outputs), 16 Analog Inputs, 256K flash memory with a 16 MHz clock speed.

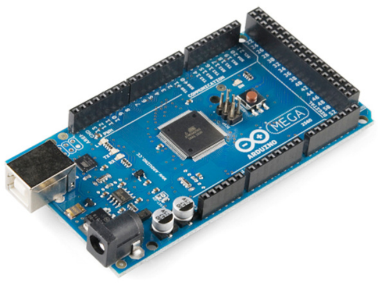
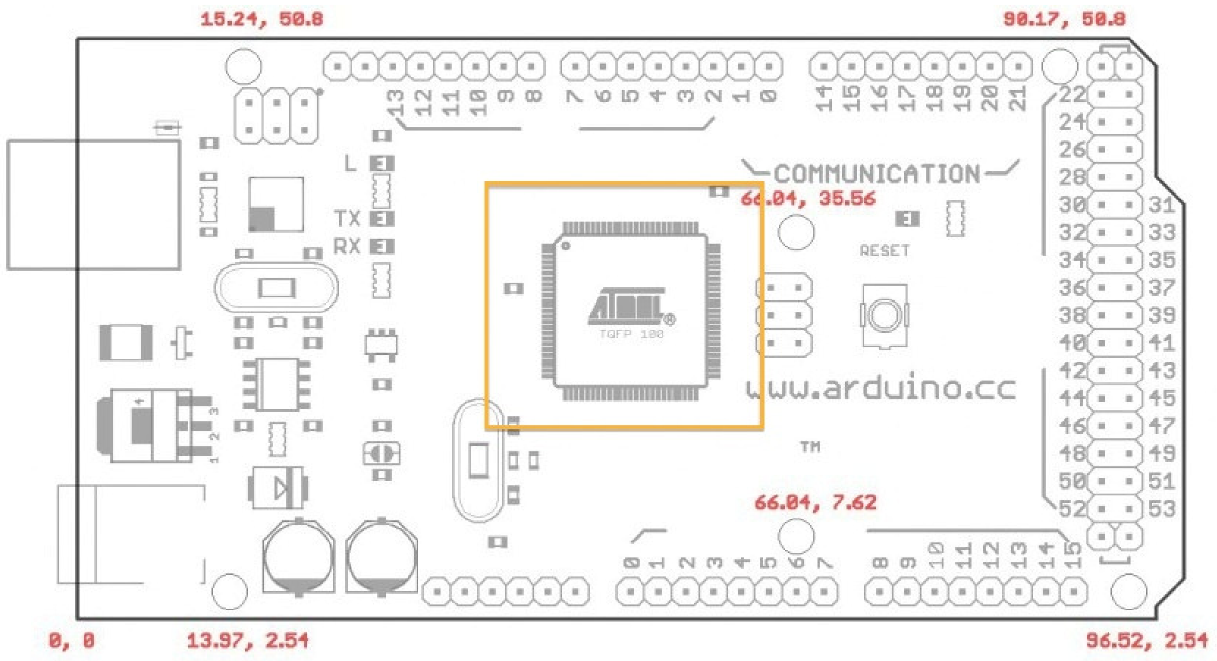
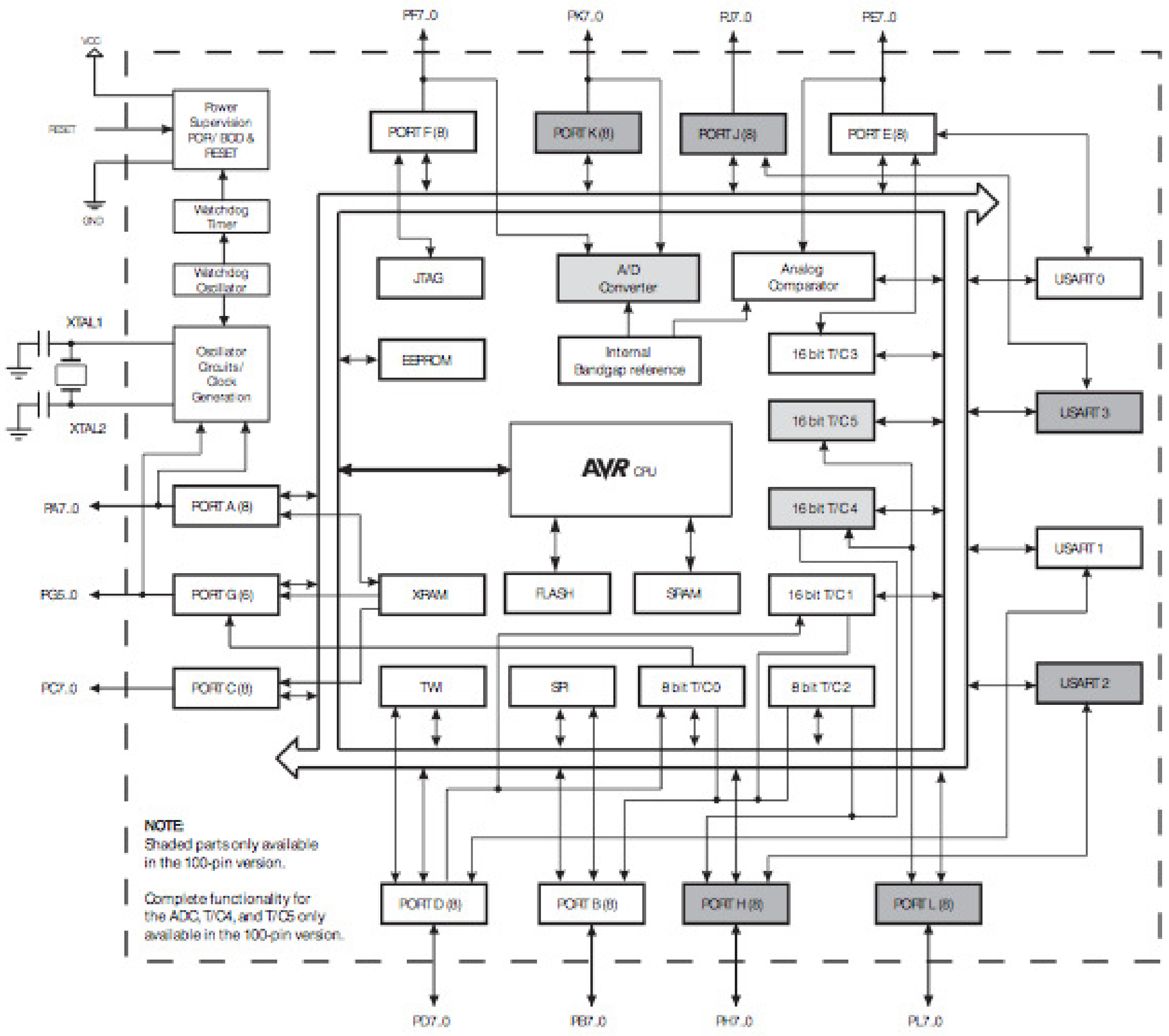


Figure 1 Arduino MEGA2560



Because this is an ongoing project there was an already working microcontroller when this year's team began working the project. The current Microcontroller being used is the ATmega2560, which is the one in the Arduino MEGA2560 board. The microcontroller is a high performance, low power 8-bit microcontroller. It has 32x8 general purpose working registers, 256K bytes of in-system self-programmable flash, 4K bytes EEPROM and 8K bytes internal SRAM. This microcontroller provides 6 sleep modes as well an internal calibrated oscillator and external and internal interrupt sources.

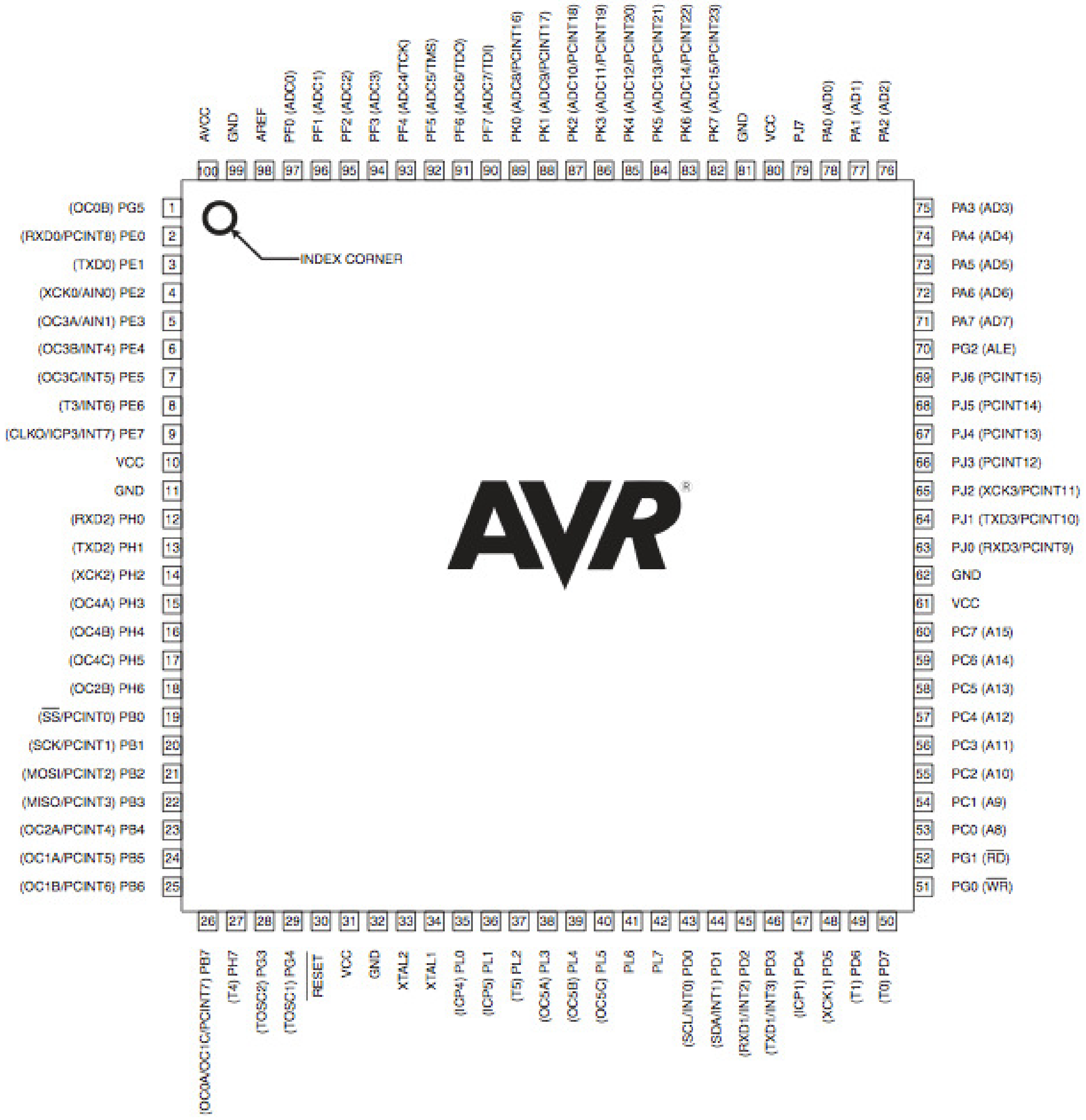
Figure 2 The internal architecture of this MCU and pin assignments are:



ATmega2560 Block Diagram

Information courtesy of the ATmega datasheet

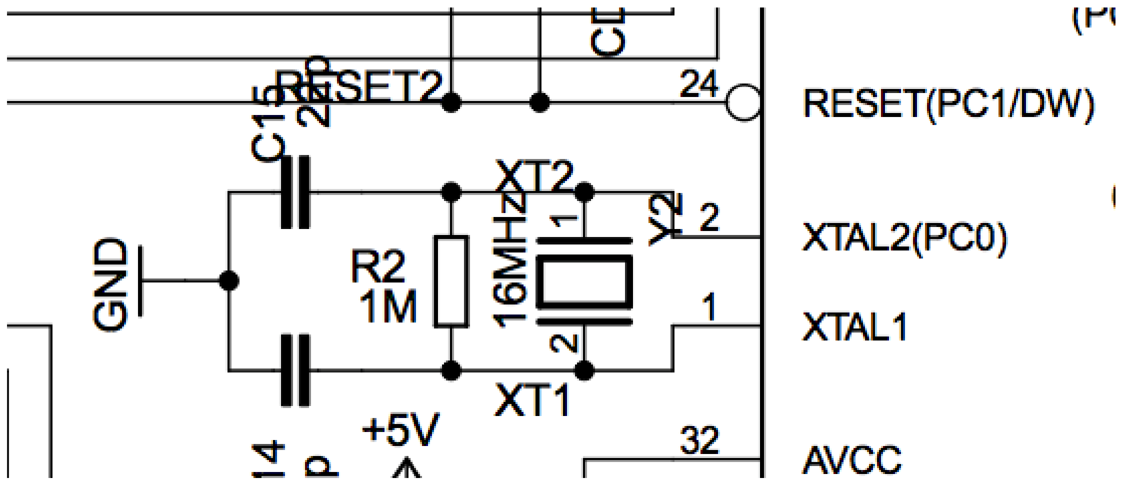
Figure 1-1. TQFP-pinout ATmega640/1280/2560



ATmega2560 pin assignments

Information courtesy of the ATmega datasheet

By default, the Internal RC Oscillator provides an approximate 16MHz clock. This clock may be selected as the system clock by programming the XTAL1 and XTAL2 pins.



The 16MHz oscillator circuit schematic connected to pins XTAL1 and XTAL2. This is the external circuitry of the board.

Information courtesy of the www.arduino.cc

3.1.2 Status of Pin Connections

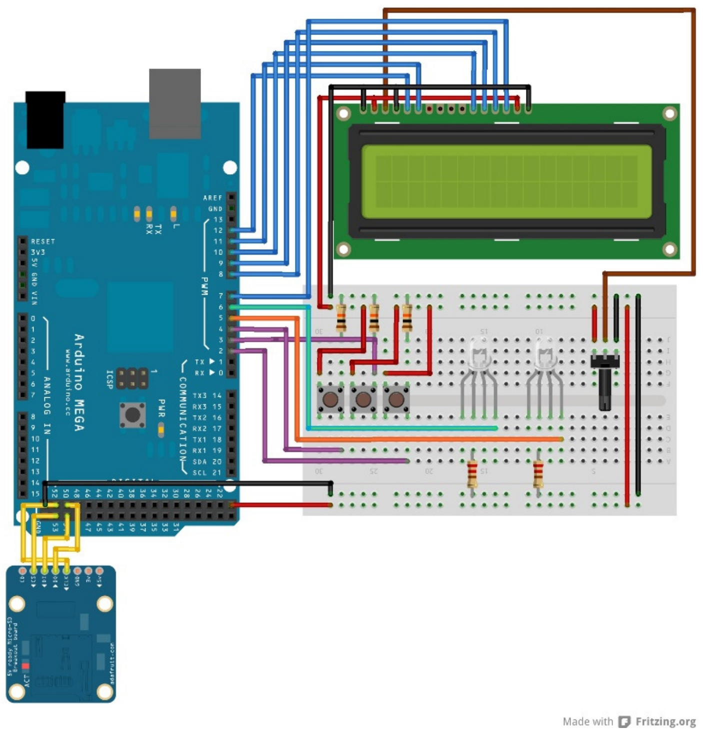


Figure 3 Pin connections from Arduino board schematic

The following pins have been selected and will be used.

Pin	Connect To	Arduino Description
15	GPS RX	TX3
14	GPS TX	RX3
12	LCD P5	
11	LCD P6	
10	LCD P11	
9	LCD P12	
8	LCD P13	
7	LCD P14	
6	LED G	Waiting/Saving data LED
5	LED R	Acquiring/Holding data LED
4	PB Save	SAVE pushbutton
3	PB Clear	CLEAR pushbutton
2	PB Toggle	Menu Toggle pushbutton
A0	Accelerometer	
A1	Battery Level	
50	Micro SD Shield D0	MISO
51	Micro SD Shield DI	MOSI
52	Micro SD Shield CLK	SCK
53	Micro SD Shield CS	SS

Turf-Tec Impact Tester Ardruino MEGA2560 Pin Assignments

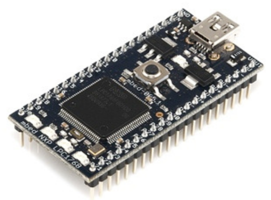
To program the pins the same code courtesy of Dr. Frank is being used this year, only pin selections have been changed. The following pin assignment breakdown is as follows:

```
            //|-----------------------------------------------------------------------------------------------|
            //|                                                                                               |
            //|    GREEN_LED_PIN, RED_LED_PIN                                   [numeric constant macros]     |
            //|                                                                                               |
            //|        These macros define the pin numbers for controlling the two application                |
            //|        LEDs (light-emitting diodes):  The green ("Acquiring/Holding Data") LED                |
            //|        and the red ("Wait/Saving Data") LED.                                                  |
            //|                                                                                               |
            //|    Used by:                                                                                   |
            //|                                                                                               |
            //|                                                                                               |
            //|        Module   Function(s)                                                                   |
            //|        ------   ------------------------------------------------------                        |
            //|        [LED]    init_LEDs()                                                                   |
            //|        [LCD]    error()                                                                       |
            //|        [PB]     process_saveButtonPress(), process_discButtonPress()                          |
            //|        [App]    loop()                                                                        |
            //|                                                                                               |
            //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

#define GREEN_LED_PIN  6
#define RED_LED_PIN    5




        //|-----------------------------------------------------------------------------------------------------|
        //|                                                                                                     |
        //|    2.5.  [Batt] Battery-level measurement constants.                          [code subsection]     |
        //|                                                                                                     |
        //|        This section defines macros for various constant parameters of the battery                   |
        //|        voltage level measurement facility.                                                          |
        //|                                                                                                     |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            //  BATT_PIN - Pin number for the battery-level measurement.
            //    USED BY:  [Batt] init_batt(), disp_blife().

#define  BATT_PIN  A1


        //|------------------------------------------------------------------------------------------------------|
        //|    2.11.  [Acc] Accelerometer interface constants.                              [code subsection]    |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            //  ACCEL_ANALOG_PIN - Pin number used for (analog) input from accelerometer.
            //  Currently, analog pin #0 is used to read accelerometer value.
            //  (This does not need to be specified as 'A0' b/c it is used as an argument to analogRead().)
            //  Used By:  [Acc] impact_detected(), accel_gs().

#define  ACCEL_ANALOG_PIN    0

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const int  accelPin    =  ACCEL_ANALOG_PIN;      // Input signal from accelerometer interface.

            // ACCEL_PIN - Same as ACCEL_ANALOG_PIN; but use this version instead when you're
            // trying to use functions other than analogRead().  Used By:  [Acc] init_accel().

#define  ACCEL_PIN           A0
```

```
        //|--------------------------------------------------------------------------------------------------------|
        //|    2.12.  [PB] Pushbutton interface constants.                                     [code subsection]    |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

#define  PUSHBUTTON_DEBOUNCE_MILLIS  150
        // Debounce interval for pushbuttons - milliseconds length of refractory period.
        // If this is too small, then the timezone-setting feature will appear to "skip" time zones
        // when the button bounces.  If this is too large, then the user can't adjust timezone very quickly.
        // Used By: [PB] check_save_btn(), check_disc_btn().

            // Pin identifiers for the save/discard pushbuttons.
            // We currently use two analog pins in digital mode.
            // Used By: [PB] init_buttons(), check_save_btn(), check_disc_btn().
            // Addtional button has been added to Toggle between screens. *CAD
            // We have changed the analog pins to digital pins with the upgrade to the mega. *CAD
            // new feature used by: [PB] intit_buttons(), check_togg_btn(). *CAD

#define  SAVE_BUTTON_PIN      4  // changed from A4 to 4
#define  DISCARD_BUTTON_PIN   3  // changed from A5 to 3
#define  TOGGLE               2  // Newly added button for Toggle feature

        //|--------------------------------------------------------------------------------------------------------|
        //|    3.7.  [LCD] LCD display control globals.                                      [code subsection]  |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            // First, initialize a global instance 'lcd' of the LiquidCrystal class for driving the LCD display.
            // Used By:
            //    [Batt]  disp_blife();
            //    [LCD]   init_lcd(), install_custom_glyph(), lcd_show_state(), lcd_show_labels(),
            //            lcd_print_num_padleft(), error(), disp_logo(), disp_gmax(), lcd_print_angle(),
            //            display_time(), display_lat_long(), display_timezone(), display_timezone_screen();
            //    [GPS]   init_GPS(), handle_GPGGA();
            //    [SD]    init_sd(), create_file();
            //    [PB]    process_saveButtonPress(), process_discButtonPress();
            //    [App]   loop().

//LiquidCrystal  lcd(10,11,9,8,7,6);      // These are the new assignments. We didn't incorporate the LEDs with the PWM pins
LiquidCrystal lcd(12,11,10,9,8,7);        // New, new assignments. *CAD
```

3.1.3 Status of Alternatives Considered

The microcontroller that is currently used is part of the Advanced Virtual RISC (or AVR) family, when looking into an alternative the team chose those from the Advanced RIS Machines (or ARM) family. *RISC = Reduced Instruction Set Computing.

Characteristic	mbed - LPC1768 	Maple

Microprocessor	32-bit Cortex-M3	32-bit Cortex-M3
Pins	40-pin to include analog, PWN, serial and more	39 Digital 16 Analog 15 PWM
Memory	512KB flash 64KB SRAM	128KB flash 20KB SRAM
Oscillator	100MHz	72 MHz
Power	4.5-9V input 5V USB output 3.3V output	4-12V input Supplies up to 500mA @ 3.3V
Programming	ARM RealView compile. API-driven development.	Maple IDE ARM-GCC
Extra	Supports many interfaces including USB, SPI, I2C CAN, Ethernet and serial	USB port for programming, external JTAG, integrated SPI and I2C, 7 Channels of direct memory access ad 4-channel timers

Microcontroller comparison chart
Information courtesy of www.sparkfun.com

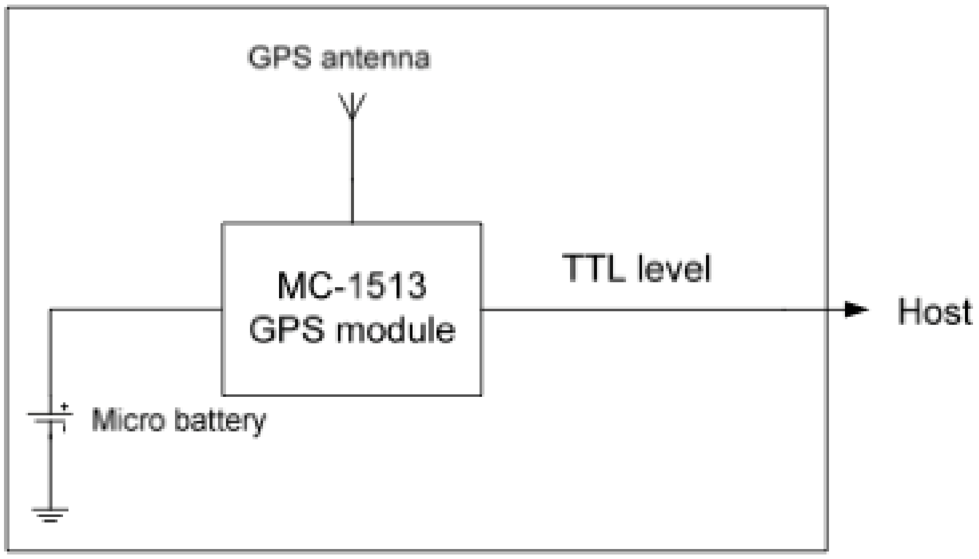
3.1.4 Overall Status

The Arduino MEGA2560 is already included in the inventory and is already incorporated in the design. All connections have been made with the other components, and the previous code has been updated in order to reflect them. The board communicates with all components properly.

3.2 GPS

3.2.1 Status of GPS

The current GPS is configured using a terminal program and ASCII NMEA packet sentences. The code provided for the GPS was provided by Dr. Frank. A new time library was added to hopefully fix the time zone errors with the GPS.



System Block Diagram
Schematic courtesy of LOCOSYS data sheet

The GPS LS20031 uses NMEA (National Marine Electronics Association) 0183 version 3.01. There are 10 interpreted sentences in NMEA for this GPS, however, the one Turf-Tec is going to use is $GPGGA which is Global Positioning System Fix Data and $GPRMC to get current date info.

A string of data is displayed in the following format:

$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx

This is an example of string of data received:

$GPGGA,053740.000,2503.6319,N,12136.0099,E,1,08,1.1,63.8,M,15.2,M,,0000*64

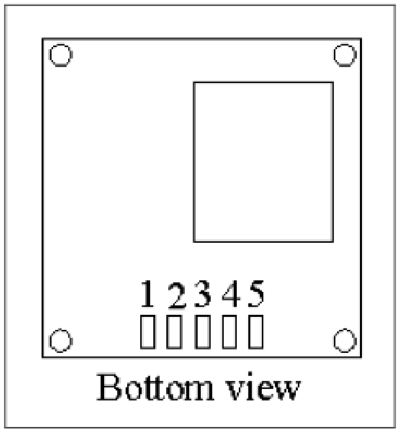
The data can be broken down with the following chart:

Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	053740.000		hhmmss.sss
Latitude	2503.6319		ddmm.mmmm
N/S indicator	N		N=north or S=south
Longitude	12136.0099		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Position Fix Indicator	1		See Table 5.1-3
Satellites Used	08		Range 0 to 12
HDOP	1.1		Horizontal Dilution of Precision
MSL Altitude	63.8	mters	
Units	M	mters	
Geoid Separation	15.2	mters	
Units	M	mters	
Age of Diff. Corr.		second	Null fields when DGPS is not used
Diff. Ref. Station ID	0000		
Checksum	*64		

$GPGGA String Breakdown
Schematic courtesy of LOCOSYS data sheet

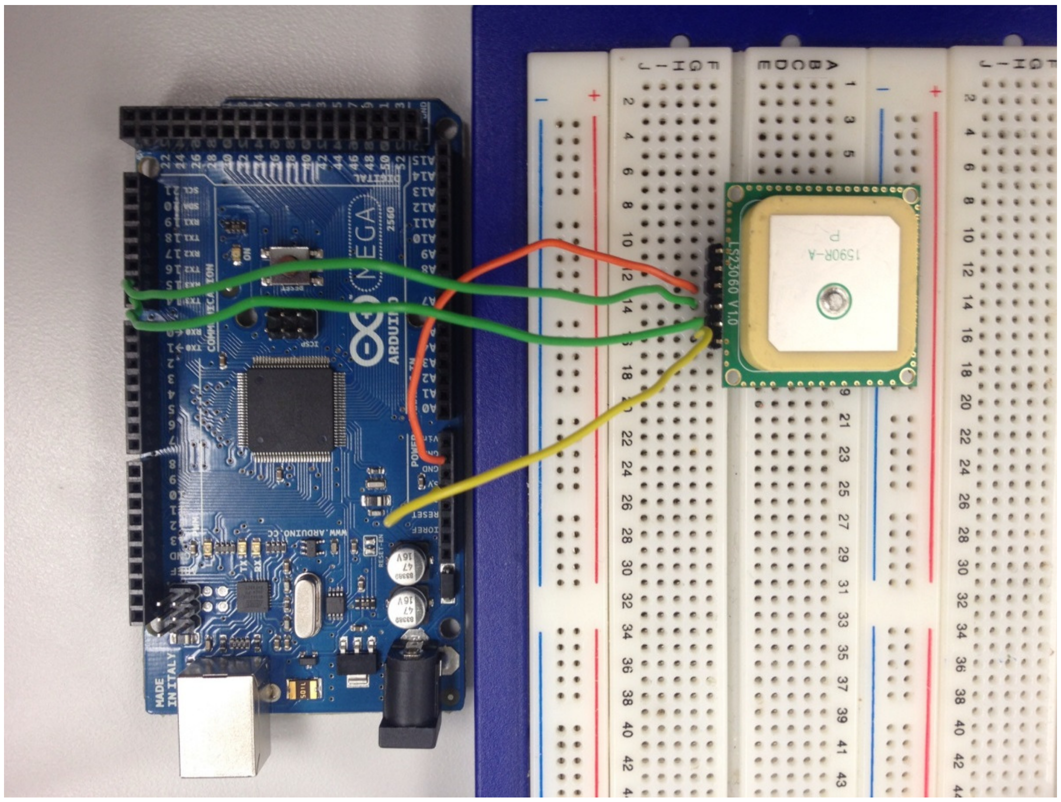
3.2.2 Status of Pin Selections

The pin assignments at this stage are the same as last year. The pins are subject to change depending on our new design. If pins need to be moved we will do so.



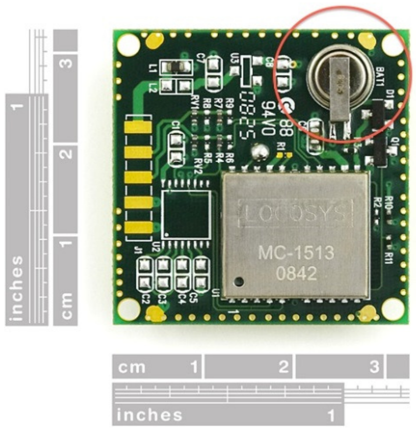
LS20031				
GPS Pin	Arduino Pin	Name	Type	Description
1	5V	VCC	P	Power input
2	15	RX	I	Data input (TTL level)
3	14	TX	O	Data output (TTL level)
4	GND	GND	P	Ground
5	GND	GND	P	Ground

GPS Pin Assignments
Schematic and information courtesy of LOCOSYS data sheet


GPS and Pin connections

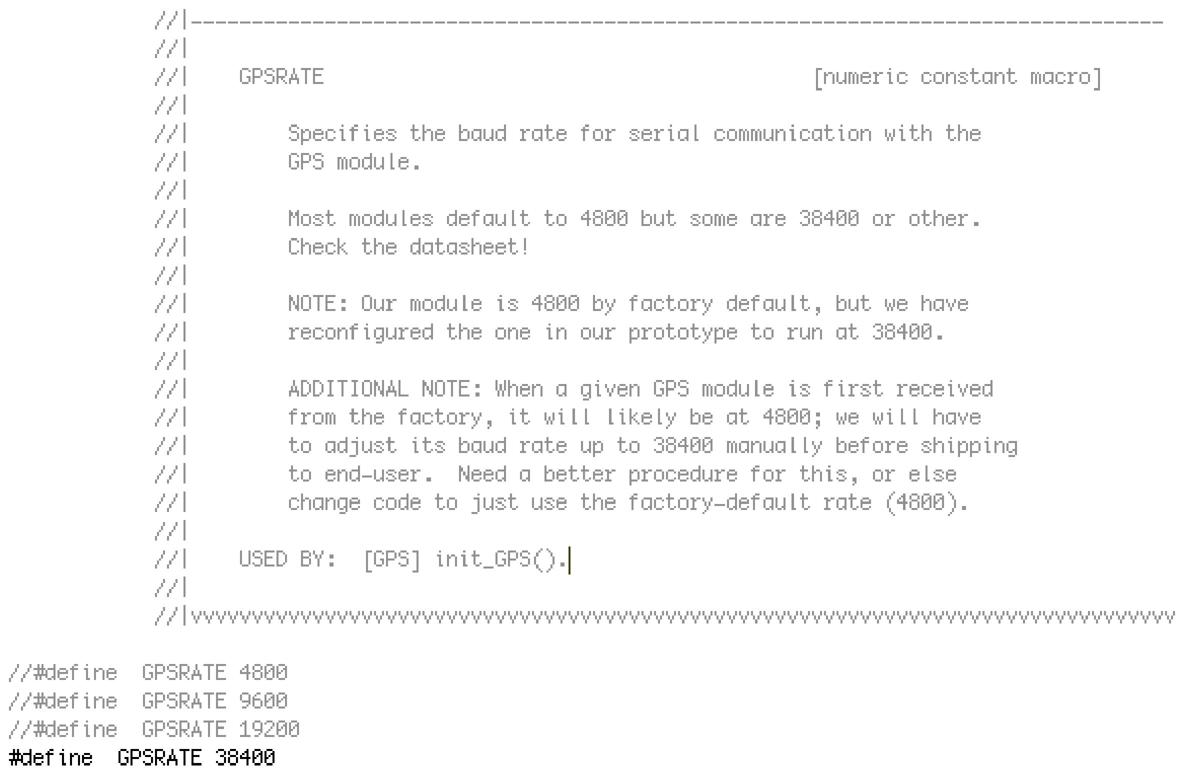
3.2.4 Status of backup battery

Last year's team decided to go with this GPS model because of its accuracy and the fact that it had a backup battery implemented already on it. This is why we are going to continue with this model as well.

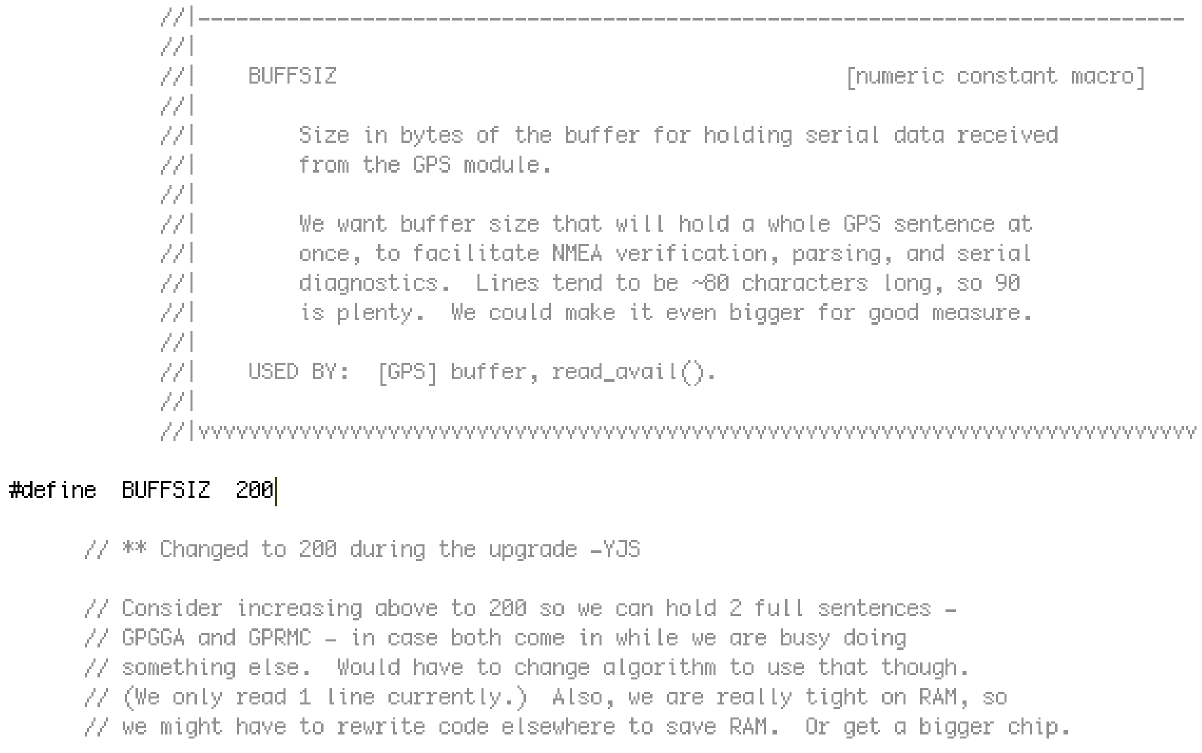


GPS backup battery implementation
Picture courtesy of www.sparkfun.com

3.2.5 Status of Connecting to the Arduino Mega2560



```
                    //|-------------------------------------------------------------------------------
                    //|
                    //|    GPSRATE                                        [numeric constant macro]
                    //|
                    //|        Specifies the baud rate for serial communication with the
                    //|        GPS module.
                    //|
                    //|        Most modules default to 4800 but some are 38400 or other.
                    //|        Check the datasheet!
                    //|
                    //|        NOTE: Our module is 4800 by factory default, but we have
                    //|        reconfigured the one in our prototype to run at 38400.
                    //|
                    //|        ADDITIONAL NOTE: When a given GPS module is first received
                    //|        from the factory, it will likely be at 4800; we will have
                    //|        to adjust its baud rate up to 38400 manually before shipping
                    //|        to end-user.  Need a better procedure for this, or else
                    //|        change code to just use the factory-default rate (4800).
                    //|
                    //|    USED BY:  [GPS] init_GPS().
                    //|
                    //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

//#define  GPSRATE 4800
//#define  GPSRATE 9600
//#define  GPSRATE 19200
#define  GPSRATE 38400
```



```
                    //|-------------------------------------------------------------------------------
                    //|
                    //|    BUFFSIZ                                        [numeric constant macro]
                    //|
                    //|        Size in bytes of the buffer for holding serial data received
                    //|        from the GPS module.
                    //|
                    //|        We want buffer size that will hold a whole GPS sentence at
                    //|        once, to facilitate NMEA verification, parsing, and serial
                    //|        diagnostics.  Lines tend to be ~80 characters long, so 90
                    //|        is plenty.  We could make it even bigger for good measure.
                    //|
                    //|    USED BY:  [GPS] buffer, read_avail().
                    //|
                    //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

#define  BUFFSIZ  200

      // ** Changed to 200 during the upgrade -YJS

      // Consider increasing above to 200 so we can hold 2 full sentences -
      // GPGGA and GPRMC - in case both come in while we are busy doing
      // something else.  Would have to change algorithm to use that though.
      // (We only read 1 line currently.)  Also, we are really tight on RAM, so
      // we might have to rewrite code elsewhere to save RAM.  Or get a bigger chip.
```

GPS global variables are then declared to include minutes, seconds, month, day, year latitude and longitude.

```
        //|------------------------------------------------------------------------------------------------------------|
        //|    3.8.  [GPS] GPS globals.                                                         [code subsection]  |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            // Create global object for managing serial communication with the GPS module.
            // Use pins 2 and 3 to talk to the GPS. 2 is the TX pin, 3 is the RX pin.
            // Used By: [GPS] read_avail(), gps_config_msg_types(), gps_set_baudrate(), incoming_GPS_data(), init_GPS().

#if ARDUINO >= 100
  SoftwareSerial  gpsSerial = SoftwareSerial(2, 3);
#else
  NewSoftSerial   gpsSerial = NewSoftSerial(2, 3);
#endif

            // Here is the buffer for storing a line of GPS data being read, until we
            // finish reading and processing it.  It comes with an associated index that
            // keeps track of where we are currently filling in characters.
            // Used By: [GPS] verify_checksum(), read_avail(), handle_GPGGA(), handle_GPRMC(), process_line(); [App] loop().

char     buffer[BUFFSIZ];   // string buffer for the sentence
char     buf_index = 0;     // index into buffer.  Used By: [GPS] read_avail().

//            // The time at which we finished receiving the previous GPS sentence.
//            // (We're not using this currently; we used to use it to know when
//            // to start looking for the next sentence.)
//
//unsigned long  last_GPS_millis;

            // Most recent GPS data obtained.
            // The time, date, location data, etc. (not all are needed)

                //  Most recent raw GPS time data - UTC/GMT; not yet timezone-adjusted.
                //  Used By: [Time] set_system_time(); [GPS] handle_GPRMC().

uint8_t    gpsHour, gpsMinute, gpsSecond, gpsMonth, gpsDay;
uint16_t   gpsYear;

                //  Most recent GPS location data.  Angle format: Integer, decimal DDDMMMMMM, where D=degrees, M=minutes*10,000
                //  (This fits in 32 bits, since 179,599,999 < MAXINT for 32-bit unsigned integers.)
                //  NOTE: Currently we do not attempt to do any smoothing/averaging of location readings.
                //  Used By: [LCD] display_lat_long(); [GPS] handle_GPRMC(); [SD] sd_save_data().

uint32_t   latitude = 0, longitude = 0;    // Default both of these to 0 (in ocean South of Ghana!)
char       latdir = 'N', longdir = 'E';    // Arbitrary default - corresponds to 0 being 'positive'.

//uint8_t    groundspeed, trackangle;    // not used

        // Date/time of "the" last (or held) GPS reading, timezone-corrected. (Should we move this to the [Time] module?)
        // Used By: [LCD] display_time(), [SD] sd_save_data().

uint8_t    theHour, theMinute, theSecond, theMonth, theDay;
uint16_t   theYear;
boolean    thePM;    // TRUE if "the" time (of last or held GPS reading) is 12:00 noon or later.
```

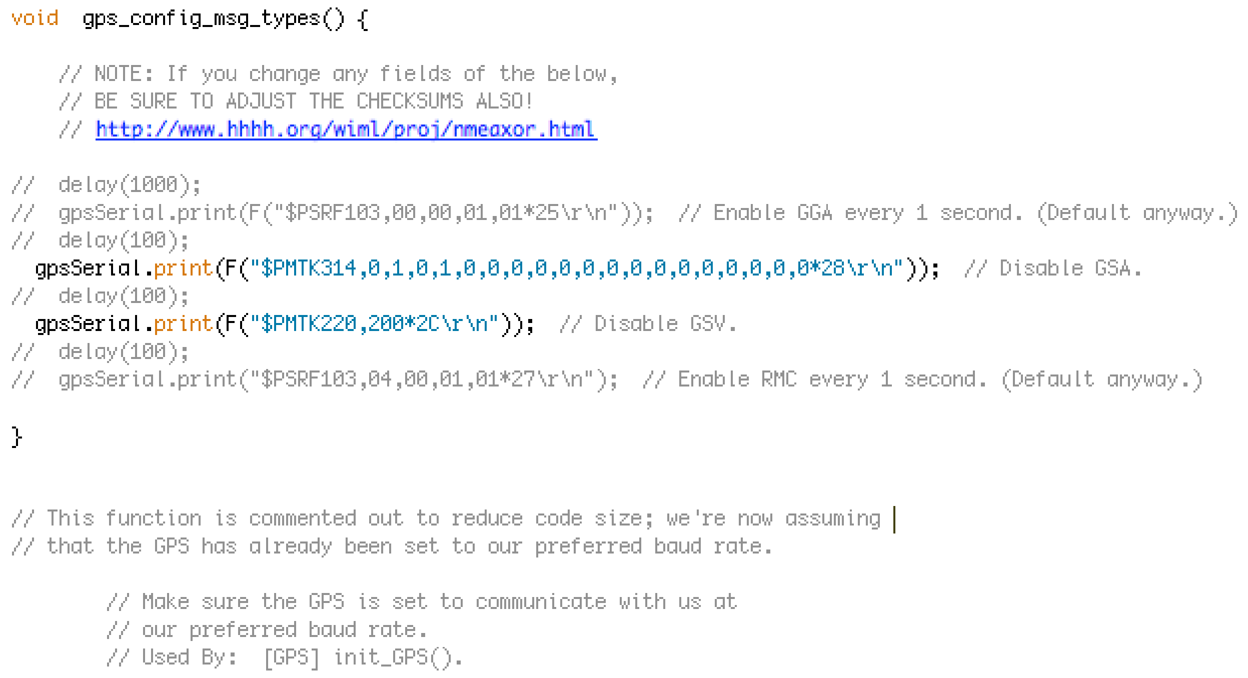
After the GPS functions are defined, this include read_avail() which reads all available characters from the GPS until the end of a line is reached, readline() which is a loop that keeps reading data until the EOL is found, init_gps() which changes the baud rate and allows for packet configuration to take place, and lastly handle_GGPA() which controls the sentences coming in. The code can be seen below:

```
        //|---------------------------------------------------------------------------|
        //|    4.8.  [GPS] GPS functions.                         [code subsection]   |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            // Verify that the string in the line buffer is a complete & correct
            // NMEA sentence by verifying the checksum.  Returns true if the check-
            // sum is present and valid; false otherwise.  NOTE: Currently does not
            // actually verify the checksum, to reduce code size.
            // Used By: [App] loop().

boolean  verify_checksum() {

// The following line is commented out to save code size since read_avail() now
// ensures that the first character stored to the buffer is always a '$' anyway.
//  if (buffer[0] != '$') return false;        // No initial '$' --> reject.

  int len = strlen(buffer);                   // get length of string in buffer

    // Rather than actually verifying the checksum, this temporary hack to reduce
    // code size just makes sure there is a '*' in the right place.
  return  (buffer[len-3] == '*');

// The meat of checksum verification below is commented out to reduce code size.
//  if (buffer[len-3] != '*') return false;    // No checksum '*' --> reject.
//  uint8_t n1 = hex2int(buffer[len-2]);       // Get first nibble of hex checksum.
//  if (n1 > 15) return false;                 // Bad 1st hex digit --> reject.
//  uint8_t n2 = hex2int(buffer[len-1]);       // Get second nibble of hex checksum.
//  if (n2 > 15) return false;                // Bad 2nd hex digit --> reject.
//  uint8_t n = (n1<<4) + n2;                  // Put the nibbles together -> checksum val.
//    // OK, let's go ahead and compute the checksum of the data.
//  uint8_t sum = 0;
//  for (char *p = buffer+1; p < buffer + len - 3; p++) {
//    sum ^= (uint8_t)(*p);
//  }
//  if (sum != n) return false;                // Checksum doesn't match --> reject.
//  return true;                               // Pass that gauntlet of tests --> accept.

}


            // Reads all available characters from GPS serial connection, at least
            // until the end of a line is reached.  Returns true if a complete line
            // was found, false otherwise.
            // Used By: [GPS] readline(); [App] loop().
```

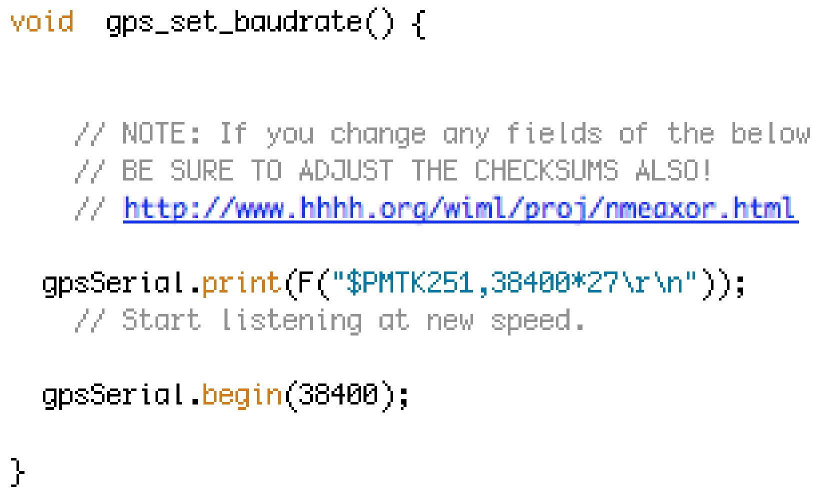
```
boolean  read_avail() {
  char     c;

  while (1) {
    c = gpsSerial.read();  // read 1 character
    if (c == -1) return false;   // no more characters available; return "line not done yet"
    //Serial.print(c);  // echo to serial - diagnostic
    if (c == '\r')          // CR character?
      continue;             // just skip over it - we use LF as line end
    if ((buf_index == 0) && (c != '$'))  // First character in buffer not a '$'?
      continue;             // Must have started in the middle of a sentence.  Ignore chars till we get a '$'.
    if ((buf_index == BUFFSIZ-1) || (c == '\n')) {  // buffer full, or line feed?
      buffer[buf_index] = 0;  // terminate string - not including newline
      buf_index = 0;  // reinitialize buf_index for next time
      return true;  // yes, we got to the end of this line
    }
    buffer[buf_index++] = c;  // add character to buffer
  }
  // never get here
}


            // Shorter version of readline() to reduce code size.
            // Used By: [GPS] init_GPS().

void  readline()  {
  while (! read_avail()) {}    // Just keep reading stuff until read_avail finds a line-end.
}




void  init_GPS()  {

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(F("Initializing GPS module..."));
#endif

  lcd.clear();
  lcd.print(F("Starting GPS..."));

  // [Starting the serial connection before turning the GPS on ensures
  // we won't miss any data from it?] <-- Moot now anyway 'cuz GPS is on all the time.

    // Connect to the GPS at the desired baud rate
  gpsSerial.begin(GPSRATE);

// The following is commented out b/c now the module is hard-wired to be "ON" all the time...
//    // Turn on GPS module
//  pinMode(GPS_PWRN_PIN, OUTPUT);
//  digitalWrite(GPS_PWRN_PIN, LOW);   // pull low to turn on!

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(F("Waiting for data from GPS..."));
#endif

  lcd.setCursor(0,1);
  lcd.print(F("Waiting for data..."));
  read_avail();             // this is all that sync_with_GPS() did, anyway

    // Now, disable the message types we're not interested in,
    //to save serial bandwidth and processing time.
    // This involves sending a couple of commands to the GPS.

  //gps_config_msg_types();

    // Confirm to the user that the GPS is online and operational.

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(F("GPS module is operating."));
#endif

  lcd.setCursor(0,2);
  lcd.print(F("GPS module running."));
  delay(LCD_READABILITY_PAUSE);  // allows user to see prev. msg before disp. is cleared

}


            // Handles receiving a GPGGA sentence from the GPS.
            // Updates # of satellites display if in normal mode.
            // Used By: [GPS] process_line().

void  handle_GPGGA() {

      //  nsats - Last number of satellites detected.  We'll read the number of satellites
      //  detected from GPGGA and display it on the LCD.  This gives the user some idea of
      //  whether he is getting a good GPS signal, and how likely the location readings are
      //  to be reasonably accurate (generally, the more satellites, the better).  It takes
      //  at least 3 satellites to get any meaningful location reading, and it takes at
      //  least 4 satellites to get an accurate reading.  (With only 3, the unit can only
      //  guess at the altitude, and if the guess is wrong, this can throw off latitude/
      //  longitude quite a bit.)

  uint8_t    nsats = 0;                   // # of satellites in use - initially 0

  char      *parseptr;   // pointer to where we are in string while parsing

  parseptr = buffer+7;  // skip "$GPGGA,"
    // skip 6 commas
  for (int ncmas = 0; ncmas < 6; parseptr++) {
    if (*parseptr == ',') ncmas++;
  }
  nsats = parsedecimal(parseptr);

#ifdef SERIAL_DIAGNOSTICS
  Serial.print(F("\tGot #satellites: "));
  Serial.println(nsats);
#endif
```

3.2.6 Status of Packet Implementation

Packet implementation will be done through the code under the function init_gps(). It will configure message types (disabling all packets except GGA and RMC) and it will set the baudrate to 38400.



```
void  gps_config_msg_types() {

    // NOTE: If you change any fields of the below,
    // BE SURE TO ADJUST THE CHECKSUMS ALSO!
    // http://www.hhhh.org/wiml/proj/nmeaxor.html

//  delay(1000);
//  gpsSerial.print(F("$PSRF103,00,00,01,01*25\r\n"));  // Enable GGA every 1 second. (Default anyway.)
//  delay(100);
  gpsSerial.print(F("$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28\r\n"));  // Disable GSA.
//  delay(100);
  gpsSerial.print(F("$PMTK220,200*2C\r\n"));  // Disable GSV.
//  delay(100);
//  gpsSerial.print("$PSRF103,04,00,01,01*27\r\n");  // Enable RMC every 1 second. (Default anyway.)

}


// This function is commented out to reduce code size; we're now assuming
// that the GPS has already been set to our preferred baud rate.

        // Make sure the GPS is set to communicate with us at
        // our preferred baud rate.
        // Used By:  [GPS] init_GPS().
```



```
void  gps_set_baudrate() {


    // NOTE: If you change any fields of the below,
    // BE SURE TO ADJUST THE CHECKSUMS ALSO!
    // http://www.hhhh.org/wiml/proj/nmeaxor.html

  gpsSerial.print(F("$PMTK251,38400*27\r\n"));
    // Start listening at new speed.

  gpsSerial.begin(38400);

}
```

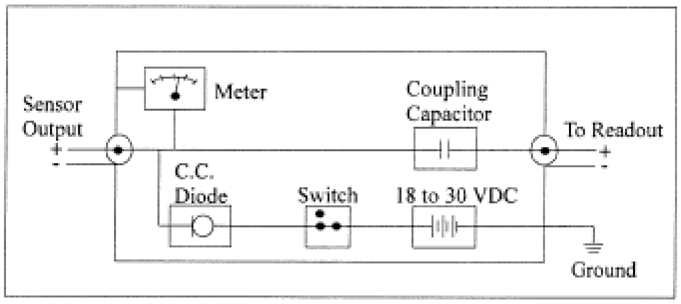
3.2.7 Overall Status

We are currently in the process of getting last year's prototype working. Once we get it working we will be able to check whether or not the GPS is working correctly. With the new library added hopefully there will not be any problems with the GPS.

3.3 Accelerometer

**All technical risks can be seen in the Risk Assessment section of this report.*

3.3.1 Status of Accelerometer



Shows the block schematic for the accelerometer.
Picture courtesy of accelerometer datasheet.

Model 352C04 is the accelerometer that was selected. It is a high performance, low power system that uses an ICP (Integrated Circuit Piezoelectric) chip that converts the vibration or the acceleration of motion on to a structure into an electrical pulse. The accelerometer provides a low amount of spectral noise when converting from vibrations to electrical pulses; it also provides a frequency range from 0.3 Hz ~15 kHz. As seen in the figure above, the accelerometer uses a special current limiting diode to limit the current specifically in the range that is acceptable for the accelerometer. The voltage regulator will be used to limit the current going into the accelerometer.

3.3.2 Status of Software Interface

For the impact to actually read GMAX values an algorithm for conversion must be written to the microcontroller to convert a voltage signal to G's. The model will convert values by dividing 10 mV/g to the output voltage that was sent to the microcontroller and then the digital signal is sent to the data logger for storage and then the user interface for viewing. We have the code for this specific conversion; it is shown below courtesy of last year's team and edited by Dr. Frank.

The code for the accelerometer breaks down as follows:

First, the interface constant global variables are declared and described in the code.

```
#define  VOLTS_PER_G         (9.92e-3)

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const  float  voltsPerGee  =  VOLTS_PER_G;    // Accerometer voltage per G-force unit.

            // GEES_PER_VOLT - Nominal Earth-gravity units per volt voltage on accelerometer output.
            // This point of this is just to avoid having to do a divide each time we convert a
            // measurement.  Using this constant, we can just do a multiply instead.  The compiler
            // will precompute it, so we don't end up doing a divide each time we use it.
            // Used By: [Acc] SCALE_FACTOR.

#define  GEES_PER_VOLT       (1.0 / VOLTS_PER_G)

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const  float  geesPerVolt  = GEES_PER_VOLT;    // G-force units per volt of accelerometer output.

            // IMPACT_THRESH - Here is the minimum threshold, in g's, for detection of the
            // start of an impact pulse.  If this is too low, then we can end up getting a
            // lot of spurious readings due to noise of various kinds.  I've tried 10 and
            // 15, but sometimes they are having noise problems, so I'm using 20 instead -
            // it seems to work pretty well.  If the threshold is too large, then the user
            // can't measure the actual Gmax of very soft surfaces.  (At some point, we
            // should perhaps add a feature to allow the user to manually adjust the
            // detection threshold to their liking; & store this preference in EEPROM.)
            // Used By: [Acc] RAW_IMPACT_THRESH.

#define  IMPACT_THRESH       (20.0)

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const  float  impactThresh  =  IMPACT_THRESH;    // Minimum detectable impact in g's.

            // RAW_IMPACT_THRESH - This is the same thing as IMPACT_THRESH above, but in raw
            // A2D-converter units, instead of in gees.  This allows faster detection of a
            // threshold crossing, because we don't have to do floating-point arithmetic on
            // each measurement.  (The compiler will precompute the value of this constant
            // at compile time.)
            // Used By: [Acc] IMPACT_DETECTED.
```

```
#define  RAW_IMPACT_THRESH   (((IMPACT_THRESH * VOLTS_PER_G) / ADC_VREF) * MAX_ADC_VAL)

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const  int  rawImpactThresh  =  RAW_IMPACT_THRESH;

            // SCALE_FACTOR - This is a compile-time precalculation to speed up conversion of
            // raw accelerometer readings to G-force values.
            // Used By: [Acc] accel_gs(), read_pulse().

#define  SCALE_FACTOR        (ADC_VREF * ONE_OVER_ADCMAX * GEES_PER_VOLT)

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const float  scaleFactor = ADC_VREF * ONE_OVER_ADCMAX * GEES_PER_VOLT;    // Precalculate ADC -> G-force scaling factor

            // The following three constants are all related to the exponential smoothing algorithm.

#define  EXP_WINDOW_SIZE   (2.0)
  //  Nominal "size" of the (fuzzy) "window" for exponential smoothing.  This "window" is nominally 2 measurements wide.
  //  Used By: [Acc] CUR_ITEM_WEIGHT.

#define  CUR_ITEM_WEIGHT   (1.0 / EXP_WINDOW_SIZE)
  //  The "weight" of the current measurement in the smoothing algorithm is the reciprocal of the nominal window size.
  //  Used By: [Acc] OLD_ITEMS_WEIGHT, accel_gs().

#define  OLD_ITEMS_WEIGHT  (1.0 - CUR_ITEM_WEIGHT)
  //  For weighted-average purposes, the combined weights of all items must sum to 1, so the cumulative weight
  //  of all measurements before the current one is given by this formula.
  //  Used By: [Acc] accel_gs().

// Commenting out const globals to save RAM - see long comment in the header of this code section
//const float  expWindowSize  = 2.0;                 // Fuzzy window size for exponential smoothing algorithm
//const float  curItemWeight  = 1.0/expWindowSize;    // Weight of newest measurement in exponential smoothing average.
//const float  oldItemsWeight = 1.0 - curItemWeight;  // Weight of older measurements in exponential smoothing average.

            // PULSE_WINDOW_MILLIS - This is the assumed maximum time interval from the
            // first threshold crossing to the impact pulse's peak.  If this is too short,
            // noise while the plunger is falling can end up being misinterpreted as a
            // false (too short) peak.  If it is too long, then each Gmax measurement
            // (real or spurious) will tie up the unit for an annoyingly long time before
            // we can do anything else (like save the value, or take another reading).
            // Used By: [Acc] read_pulse().


#define  PULSE_WINDOW_MILLIS       (500)
  // Previous values tried were 100 and 200, but they seemed to result in drop noise still
  // being picked up as an impact reading too often.  Now trying 500 (1/2 second) instead.

// Obsoleted now that we have the "HOLD" application state.
//            // REFRACTORY_PERIOD_MILLIS - Refractory period during which we can't detect
//            // any new pulses after the initial pulse detection in milliseconds - 1000 is
//            // 1 second.  The point of this is to previous subsequent "bounce" impacts
//            // from being misinterpreted as if they were additional real impacts.  We turn
//            // on the red lamp during this period to tell the user he needs to wait until
//            // it's over before attempting to save/discard and take another reading.
//            // If this is too short, a bounce could be read as another impact (but will be
//            // invalid since the plunger is falling from the wrong height).  If it's too
//            // long, it's just annoying because it ties up the unit for too long.
//
//#define  REFRACTORY_PERIOD_MILLIS  (1000)
//  // Previously we used 2000, when subsequent measurements were initiated automatically.
//  // Now we make it shorter (1000) because the unit won't take any new measurements anyway
//  // until the previous one has been (manually) either saved or discarded.
```

The accelerometer functions consist of init_accel() which declares the accelerometer pin as an input, the boolean function impact_detected() which if held true it returns the reading in earth gravity units. The raw accelerometer reading is defined by accel_gs() and is then converted. Lastly, pulses are read through the read_pulse() function. The functions can be reviewed in detailed below:

```
        //|----------------------------------------------------------------------------------------------------|
        //|    4.11.  [Acc] Accelerometer functions.                                       [code subsection]  |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            // Needed initialization pursuant to reading accelerometer values.
            // Used By: [App] setup().

void  init_accel()  {

    // Make sure analog pin 0 (accelerometer) is configured as an input.

  pinMode(ACCEL_PIN, INPUT);
      // Is this strictly needed?  Or is it the default anyway?
      // Or does analogRead() override it anyway?

//Obsoleted now that we have the HOLD state.
//    // The purpose of this is just to make sure that we can begin detecting pulses
//    // immediately.  So, make sure we're outside the refractory region.
//
//  cross_thresh_millis = millis() - PULSE_WINDOW_MILLIS;

}

            // This routine simply checks whether a probe impact has begun by seeing
            // if the raw accelerometer reading is above threshold.  If this produces
            // too many spurious readings, there are 2 options to fix this: (1) raise
            // the impact detection threshold, (2) use the smoothed reading instead
            // (call accel_gs() and compare result to impactThresh).  Raising the
            // threshold can cause soft readings to be missed; using the smoothed
            // data can slow down our response-time slightly (which could possibly
            // make us miss extremely fast pulses).  Used By: [App] loop().

boolean  impact_detected()  {

  raw_adc_val  =  analogRead(ACCEL_ANALOG_PIN);        // Raw 10-bit integer value (0-1023) from A2D converter.

  return  (raw_adc_val  >=  RAW_IMPACT_THRESH);
}
```

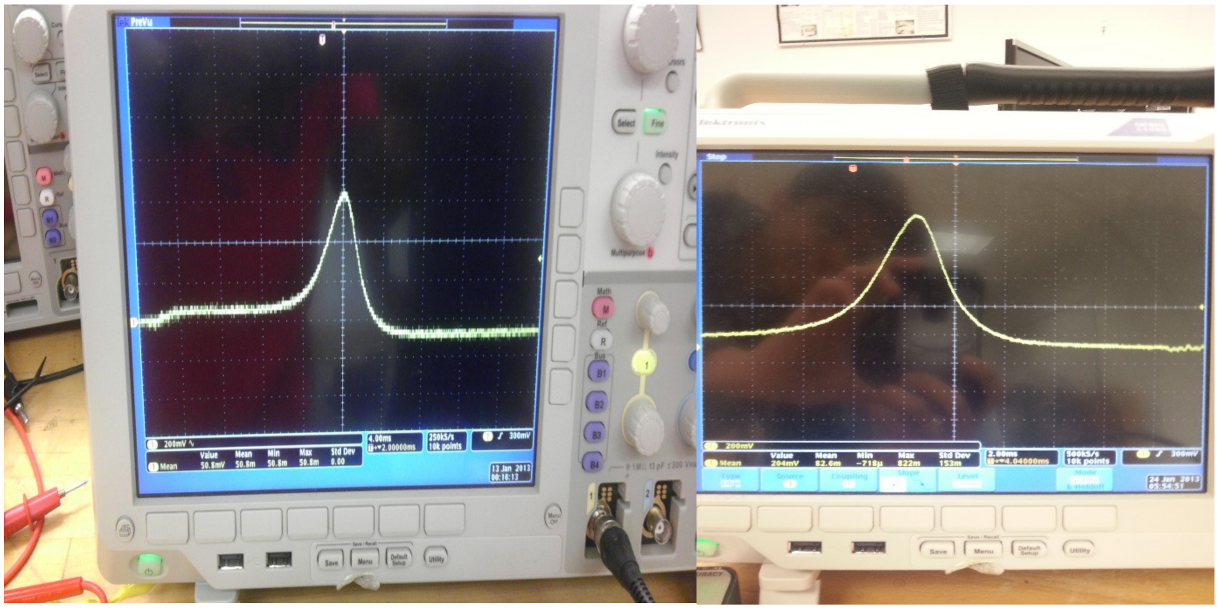
```
float  accel_gs()  {

    // Get the raw numeric reading from the A2D converter.

  int     adc_val   =  analogRead(ACCEL_ANALOG_PIN);        // Raw 10-bit integer value (0-1023) from A2D converter.

    // Convert the raw reading to the acceleration in g's.

  //Obsoleted by faster calculation below.
  //  float  volts    =  ADC_VREF*adc_val*oneOver1024; // Scale ADC reading to a voltage in volts.
  //  float  nom_acc  =  volts*GEES_PER_VOLT;           // Calculate nominal acceleration value in G-force units.

  float  nom_acc  =  adc_val * SCALE_FACTOR;          // Using this precalculated scale-factor saves us a couple
    // of floating-point multiplies. NOTE: We could make this routine even slightly faster by just working with
    // (and returning) the unscaled value.

  //Very verbose diagnostics used in debugging - now commented out.
  //#ifdef SERIAL_DIAGNOSTICS
  //  if (nom_acc > 0.0) {
  //    Serial.print("Got reading ADC = ");
  //    Serial.print(adc_val,DEC);
  //    Serial.print(", volts = ");
  //    Serial.print(volts,DEC);
  //    Serial.print(", accel = ");
  //    Serial.print(nom_acc,DEC);
  //    Serial.println(".");
  //  }
  //#endif

    // Here might be a good place to do any needed calibration corrections to convert
    // the nominal G-force value to a value comparable to that obtained by other instruments.

    // This is a simple exponential-smoothing algorithm.  Effectively it averages the last
    // several measurements (determined by expWindowSize), with an exponentially-decaying
    // weight on older measurements.  This essentially averages over about a 1/2-ms window.

  past_accel = (past_accel * OLD_ITEMS_WEIGHT) + (nom_acc * CUR_ITEM_WEIGHT);
  return  past_accel;  // Returns the smoothed acceleration value as the reading.
```

```
void  read_pulse()  {

  unsigned long elapsed_millis;

    // At this point we have no past data, except for a single reading that tells us we've
    // crossed the threshold.  The following line is a hack where we average that reading
    // together with 0, and pretend that the result constitutes our exponentially-smoothed
    // reading from our infinite history - as if the input instantly jumped from 0 to just
    // above threshold between one reading and the next.  This is inaccurate, but any
    // inaccuracy will fade away after a few more samples (within a millisecond, and
    // probably well before the peak of the pulse is reached), so it shouldn't affect the
    // final overall pulse height very much if at all.

  float  last_reading  =  raw_adc_val * SCALE_FACTOR;

  past_accel     =  last_reading / 2.0;

    // Initialize max value to first reading above threshold.
  float  max_reading = last_reading;

    // The following loop reads smoothed accelerometer values as often as possible
    // over the next few hundred ms... This is assumed to be long enough to capture
    // the true peak of the pulse, while being a short enough to appear responsive
    // and not to interfere too much with other tasks (such as updating the time
    // display).

  do {

    last_reading = accel_gs();

    // Update max value as needed.

    if (last_reading > max_reading) {
      max_reading = last_reading;
    }

    // Next, calculate the time elapsed since we first crossed the threshold.

    unsigned long now_millis = millis();
    elapsed_millis = now_millis - cross_thresh_millis;


  } while (elapsed_millis < PULSE_WINDOW_MILLIS);    // Continue until enough time has passed

//This is commented out because it is superseded by the "HOLD" application state.
//  // Enter the "refractory" state where we ignore all accelerometer data for
//  // a couple of seconds while waiting for subsequent bouncing of the probe to stop.
//
//  enter_refractory_state();

  // At this point, max_reading is the Gmax value (max height of the pulse).
  // Display it on the LCD display.  Also saves it in the "lastGmaxRound" global var.

  disp_gmax(max_reading);    // Display it on the LCD display.

  // Also print it to the serial port, for diagnostic purposes.

#ifdef SERIAL_DIAGNOSTICS
  Serial.print(F("Gmax = "));
  Serial.print(max_reading);
  Serial.println('.');
#endif

} // end read_pulse().
```

3.3.3 Status of ICP (Integrated Circuit Piezoelectric)

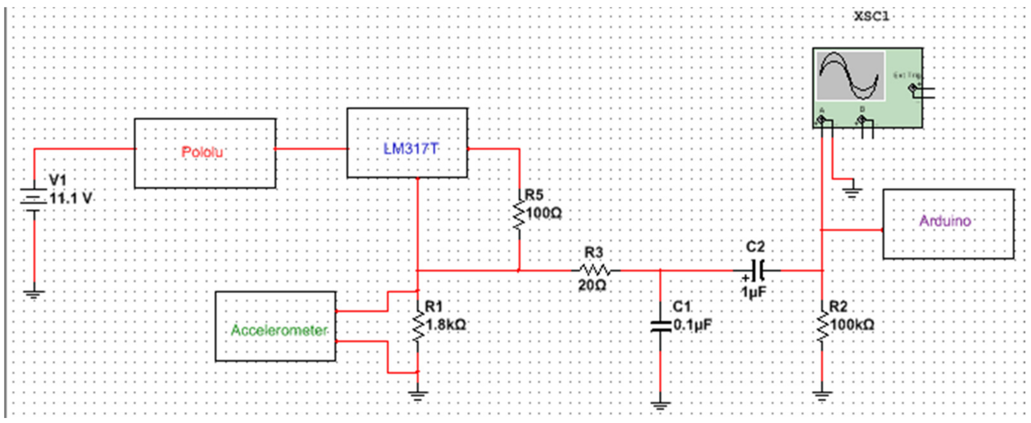
The ICP chip that is in the accelerometer is used to convert the vibration into an electrical pulse. The waveform or pulse that will be output from the accelerometer can appear like the one in figure below taken by last year's team. From this figure one can notice a lot of noise in the output signal. In order to reduce such noise, one can implement a low pass filter that can cut off those high frequency glitches. Given in equation (1) shows how we calculated the cutoff frequency for the low pass filter that we are currently using. Another way could be to include a plastic material or Teflon spray (non-sticking substance) in the hole where the missile is dropped to also reduce the noise from the missile sliding down the edges of the hole.

$$\boldsymbol{cutoff\ frequency} = \omega_C = \frac{1}{RC} = \frac{1}{(20)(0.1\mu F)} = \mathbf{500\ kHz} \qquad (\mathbf{1})$$



Shows an actual example of an output signal that accelerometer generates

To ensure that an output signal is really being produced, the circuit shown in figure 5 was implemented by last year's team; the circuit shows the new battery going straight into the Pololu (which is the boost converter) and from there into the LM317T, which is the current source in this case with a 1.8kΩ resistor used to cut the accelerometer current down to 4 mA. The expression to cut the current down is given below in equation (2). The accelerometer is then connected to the resistance. Then after a reading, the signal will then pass through the bandpass and will filter any high frequency glitches and also will be used as a DC blocker.



Shows the implemented circuit used to test the accelerometer

The current was changed from 13 mA to 4mA because Mr.Mascaro made a request. Where V_a is the DC voltage bias being outputted from the accelerometer, I_C being the current source that was developed, I_a being the accelerometer current and R_L being the load resistance that cuts the current of the accelerometer.

$$\boldsymbol{R_L} = \frac{V_a}{I_c - I_a} = \frac{10.6\ V}{13\ mA - 4\ mA} = \mathbf{1.8}\ \boldsymbol{k\Omega} \qquad (\mathbf{2})$$

3.3.4 Status of Alternatives Considered

Characteristics	**Model 352C04**	**Model 052A60**
Size	22.4 mm x 11.2 mm	21.6 mm x 9.53 mm
Power Requirements	18 ~ 30V 2mA ~ 20mA	18 ~ 30V 2mA ~ 20mA
Frequency Range	0.3 Hz ~ 15kHz	5 Hz ~60 kHz
Weight	5.8g	6g
Spectral noise (1kHz)	39 (µm/sec^2)/√Hz	147 (µm/sec^2)/√Hz
Protocol	General Purpose	High Frequency

Accelerometer Comparison Chart
Courtesy of www.pcb.com

As you can see from the comparison chart from above, both are fairly similar but that they both can get the job done but model 352C04, which was selected, is more suitable for what we want to implement.

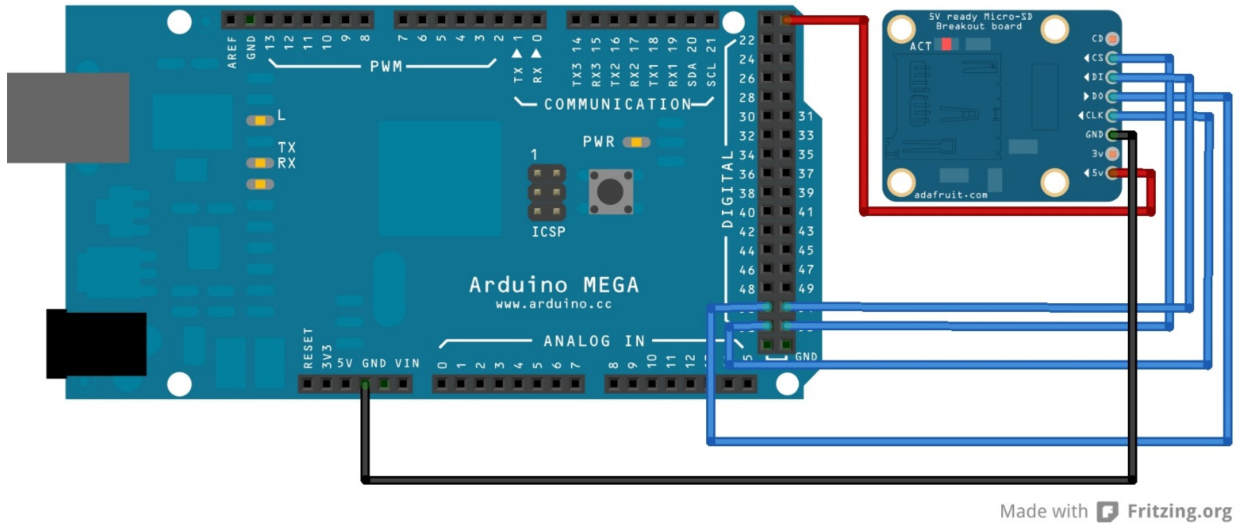
3.3.5 Overall Status

As of right now the boost converter is outputting the correct voltage for the accelerometer which is about 18.5V. We have not tested the accelerometer itself but we know that last year's team was able to get accurate readings.

3.4 Data Logger

**All technical risks can be seen in the Risk Assessment section of this report.*

3.4.1 Status of Data Logger



MicroSD card shield connected to the Arduino MEGA

The data logger component selected is a 4GB micro SD card and shield. The micro SD shield operates using 3.3V and the current draw when writing to the card can be up to 100mA. The micro-SD shield will connect directly to pins 3.3V, GND, 50, 51, 52 and 53 located on the Arduino Mega board. Communication with the Arduino Mega Board is as follow can found in the following section.

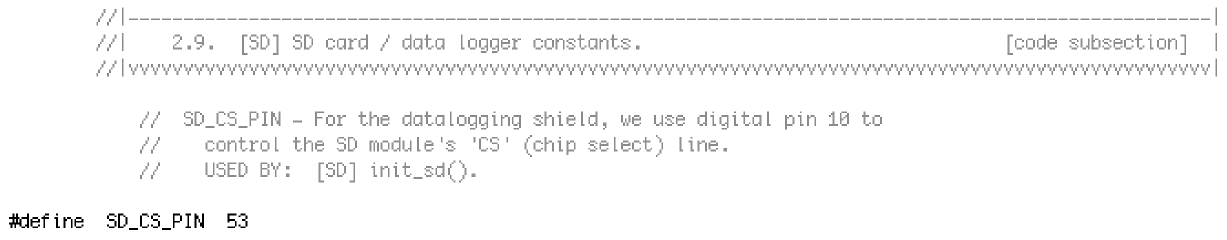
3.4.2 Status of Connection to the Ardunio Mega

The Micro SD Shield will be connected directly to the Arduino Mega board with wires using the following pin connections:

- Connect the 5V pin to the 5V pin on the Arduino

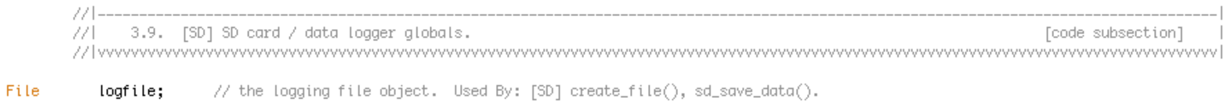
- GND pin to the GND pin on the Arduino
- Connect CLK to pin 52
- Connect DO to pin 50
- Connect DI to pin 51
- Connect CS to pin 53
-

Code for the data logger begins with pin selection for the chip select pin.



```
//|-------------------------------------------------------------------------------------------------|
//|    2.9.  [SD] SD card / data logger constants.                              [code subsection]  |
//|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

    //  SD_CS_PIN - For the datalogging shield, we use digital pin 10 to
    //    control the SD module's 'CS' (chip select) line.
    //    USED BY:  [SD] init_sd().

#define  SD_CS_PIN  53
```

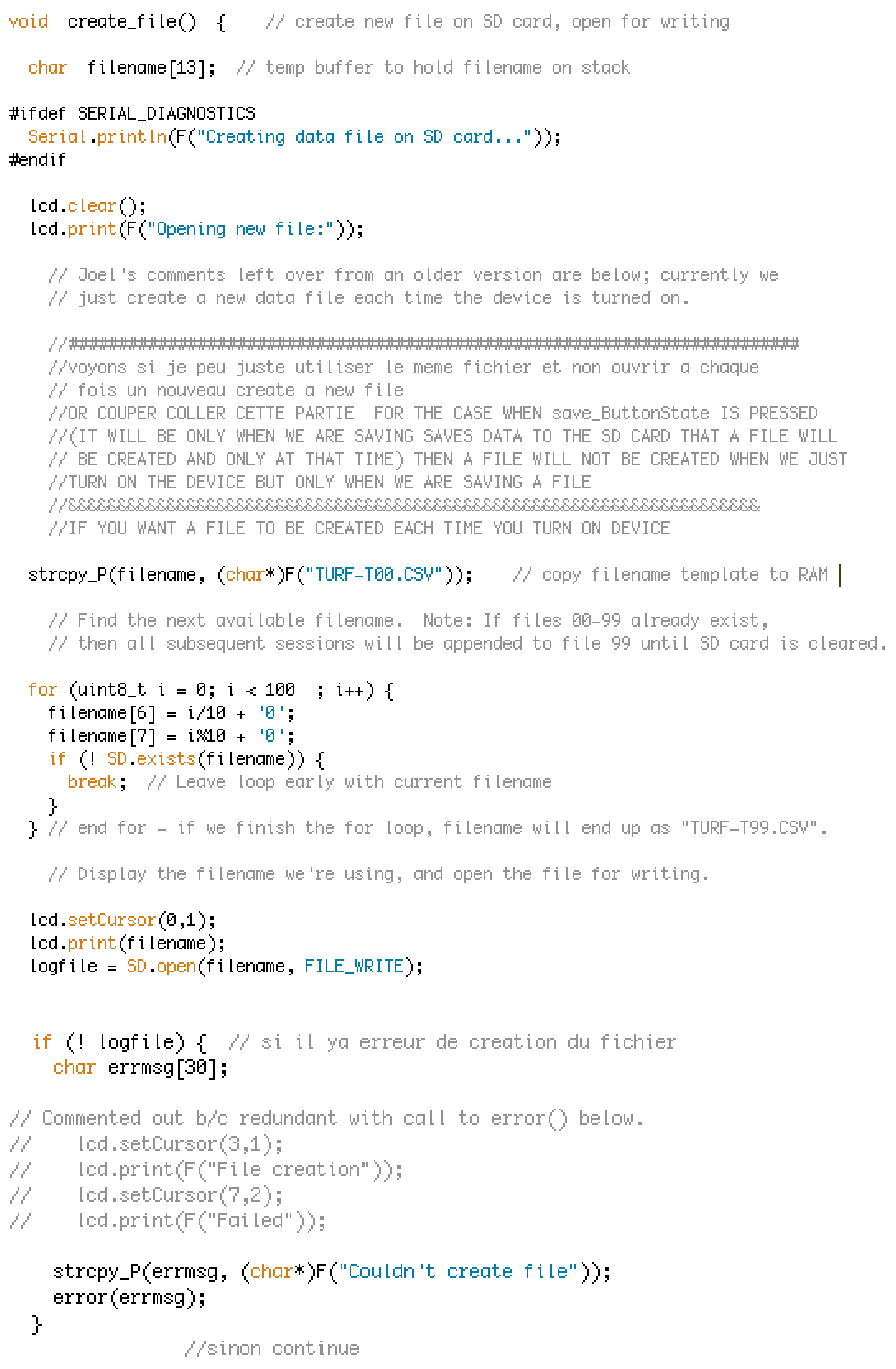
Global variable logfile is created to create the files in which all the data will be stored into.



```
//|----------------------------------------------------------------------------------------------------------------------|
//|    3.9.  [SD] SD card / data logger globals.                                                      [code subsection]    |
//|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

File      logfile;      // the logging file object.  Used By: [SD] create_file(), sd_save_data().
```

The SD card functions are as follows. The init_sd() function initializes the card and enables it. The code checks that the CS pin is set to output, if card can't be initialized, an error will show. If no error, the create_file() function will run and create a file on the SD card labeled "TURF-TXX.CSV"- the XX replaced with the corresponding file number.

```
        //|------------------------------------------------------------------------------|
        //|    4.9.  [SD] SD card functions.                            [code subsection]  |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

            // Initialize SD card drive.  Used By: [App] setup().

void  init_sd()  {    // assumes LCD already initialized

  char errmsg[30];

  // initialize the SD card
#ifdef SERIAL_DIAGNOSTICS
  Serial.print(F("Initializing SD card..."));
#endif

  lcd.clear();
  lcd.print(F("Enabling SD card..."));

    // make sure that the default chip select pin is set to
    // output, even if you don't use it:
  pinMode(SD_CS_PIN, OUTPUT);  //##pour le chipset select the la sd card

    // see if the card is present and can be initialized:
  if (!SD.begin(SD_CS_PIN)) //## sd card do not initialize
  {

// Commented out b/c redundant with call to error() below.
//    lcd.setCursor(4,2);
//    lcd.print(F("Card failed!"));

    strcpy_P(errmsg, (char*)F("Bad/missing SD card"));
    error(errmsg);
  }

    // Give user a visual confirmation that the SD initialized properly.

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(F("SD card initialized.")); //##sd card do initializes correctly
#endif

  lcd.setCursor(0,1);
  lcd.print(F("SD card activated."));
  delay(LCD_READABILITY_PAUSE);
```



```
void  create_file()  {    // create new file on SD card, open for writing

  char  filename[13];  // temp buffer to hold filename on stack

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(F("Creating data file on SD card..."));
#endif

  lcd.clear();
  lcd.print(F("Opening new file:"));

    // Joel's comments left over from an older version are below; currently we
    // just create a new data file each time the device is turned on.

    //##########################################################################
    //voyons si je peu juste utiliser le meme fichier et non ouvrir a chaque
    // fois un nouveau create a new file
    //OR COUPER COLLER CETTE PARTIE  FOR THE CASE WHEN save_ButtonState IS PRESSED
    //(IT WILL BE ONLY WHEN WE ARE SAVING SAVES DATA TO THE SD CARD THAT A FILE WILL
    // BE CREATED AND ONLY AT THAT TIME) THEN A FILE WILL NOT BE CREATED WHEN WE JUST
    //TURN ON THE DEVICE BUT ONLY WHEN WE ARE SAVING A FILE
    //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
    //IF YOU WANT A FILE TO BE CREATED EACH TIME YOU TURN ON DEVICE

  strcpy_P(filename, (char*)F("TURF-T00.CSV"));    // copy filename template to RAM

    // Find the next available filename.  Note: If files 00-99 already exist,
    // then all subsequent sessions will be appended to file 99 until SD card is cleared.

  for (uint8_t i = 0; i < 100  ; i++) {
    filename[6] = i/10 + '0';
    filename[7] = i%10 + '0';
    if (! SD.exists(filename)) {
      break;  // Leave loop early with current filename
    }
  } // end for - if we finish the for loop, filename will end up as "TURF-T99.CSV".

    // Display the filename we're using, and open the file for writing.

  lcd.setCursor(0,1);
  lcd.print(filename);
  logfile = SD.open(filename, FILE_WRITE);


  if (! logfile) {  // si il ya erreur de creation du fichier
    char errmsg[30];

// Commented out b/c redundant with call to error() below.
//    lcd.setCursor(3,1);
//    lcd.print(F("File creation"));
//    lcd.setCursor(7,2);
//    lcd.print(F("Failed"));

    strcpy_P(errmsg, (char*)F("Couldn't create file"));
    error(errmsg);
  }
                //sinon continue
```

The code for the SD card also converts the GPS data and formats it to display the day, time, GMAX, latitude and longitude respectively in an organized matter. This is found in function log_angle() and sd_save_data().

```
#ifdef SERIAL_DIAGNOSTICS
  Serial.print(F("Logging to: "));
  Serial.println(filename);
#endif

  lcd.setCursor(0,2);
  lcd.print(F("Writing header..."));

  //printing the heading on the csv file
//Commented out Turf-Tec address to reduce code size.
//  logfile.println(F("   Turf-Tec International"));       // SENDS HEATHER TO THE FILE
//  logfile.println(F(" 1471 Capital Circle NW,  # 13"));
//  logfile.println(F("  Tallahassee, FL.  32303 USA"));
//  logfile.println(F("   Order Line (800) 258-7477"));
//  logfile.println(F("    Phone 01 (850) 580-4026"));
//  logfile.println(F("     Fax 01 (850) 580-4027"));
  logfile.println(F("DAY,TIME,GMAX,LATITUDE,LONGITUDE"));

#ifdef SERIAL_DIAGNOSTICS
//  Serial.println(F("   Turf-Tec International"));
//  Serial.println(F(" 1471 Capital Circle NW,  # 13"));
//  Serial.println(F("  Tallahassee, FL.  32303 USA"));
//  Serial.println(F("   Order Line (800) 258-7477"));
//  Serial.println(F("    Phone 01 (850) 580-4026"));
//  Serial.println(F("     Fax 01 (850) 580-4027"));
  Serial.println(F("DAY,TIME,GMAX,LATITUDE,LONGITUDE"));  //juste pour imprimer le heading sur le serial
#endif //SERIAL_DIAGNOSTICS

  logfile.flush();    // ensure header data actually gets written right now (in case we power-off soon)

  lcd.setCursor(0,3);
  lcd.print(F("File ready for data."));
  delay(LCD_READABILITY_PAUSE);
}

            // Write lat/long coordinates to the data file on the SD card.
            // The output format can be copy & pasted into Google Earth or Google Maps.
            // Input format:  DDDMMmmmm (DDD degrees, MM.mmmm minutes)
            // Output format: +/-DD.ddddddd (DDD.ddddddd degrees)
            // Used By: [SD] sd_save_data().
```

```
void  log_angle(uint32_t angle, char dir) {

  unsigned int degs = angle/1000000;                   // Extract integer degrees field (DDD) from angle
  float mins_float = (angle%1000000)/10000.0;          // Extract integer ten-thousanths-of-a-minute field (MMmmmm) from angle,
  float degs_float = float(degs) + (mins_float/60.0);  // Put degrees and minutes together to make floating degrees DDD.ddddddd
  if (dir == 'W' || dir == 'S') degs_float *= -1;      // Attach minus sign to West or South coordinates.
  logfile.print(degs_float, DEC);                      // Print all significant figures to log file.

#ifdef SERIAL_DIAGNOSTICS
  Serial.print(degs_float, DEC);
#endif

}

            // Saves all current data to a new record (line) in the output .csv file
            // on the SD card.  Used By: [PB] process_saveButtonPress().

void sd_save_data() {

  digitalWrite(RED_LED_PIN, LED_ON);  // turn RED LED on while saving data
  delay(800);

    // log the DATE in US (month/day/year) format

  logfile.print(theMonth, DEC);
  logfile.print('/');
  logfile.print(theDay, DEC);
  logfile.print('/');
  logfile.print(theYear, DEC);
  logfile.print(',');

    // log the TIME (12-hour format)


  logfile.print(theHour, DEC);
  logfile.print(':');
  logfile.print(theMinute, DEC);
  logfile.print(':');
  logfile.print(theSecond, DEC);
  if  (thePM)  {
    logfile.print(F(" PM"));
  } else {
    logfile.print(F(" AM"));
  }

  logfile.print(',');

#ifdef SERIAL_DIAGNOSTICS
  Serial.print(theDay, DEC);
  Serial.print('/');
  Serial.print(theMonth, DEC);
  Serial.print('/');
  Serial.print(theYear, DEC);
  Serial.print(F(", "));
  Serial.print(theHour, DEC);
  Serial.print(':');
  Serial.print(theMinute, DEC);
  Serial.print(':');
  Serial.print(theSecond, DEC);
  if  (thePM)  {
    Serial.print(F(" PM"));
  } else {
    Serial.print(F(" AM"));
  }
  Serial.print(',');
#endif

    //########################################
    //GMAX VALUE
    //logging the Gmax on the file

  logfile.print(lastGmaxRound, DEC);
  logfile.print(',');
```

```
#ifdef SERIAL_DIAGNOSTICS
  Serial.print(lastGmaxRound, DEC);
  Serial.print(',');
#endif

    //############################################
    //  Log the latitude/longitude as signed decimal degrees
    //  (This allows direct copy&paste of coordinates into Google Maps or Google Earth.)

  log_angle(latitude, latdir);

  logfile.print(',');

#ifdef SERIAL_DIAGNOSTICS
  Serial.print(',');
#endif

  log_angle(longitude, longdir);

  logfile.println(); //to rap back the the next record
  logfile.flush();    // Ensures data actually gets written to the file now.

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(); //to rap back the the next record
  Serial.flush();
  Serial.println("Entry saved on file !!! ");
#endif

    //END OF SENDING DATA ("DAY,TIME,GMAX VALUE,LONGITUDE,LATITUDE") TO THE SD CARD
```

3.4.3 Status of Communicating with the USB Port

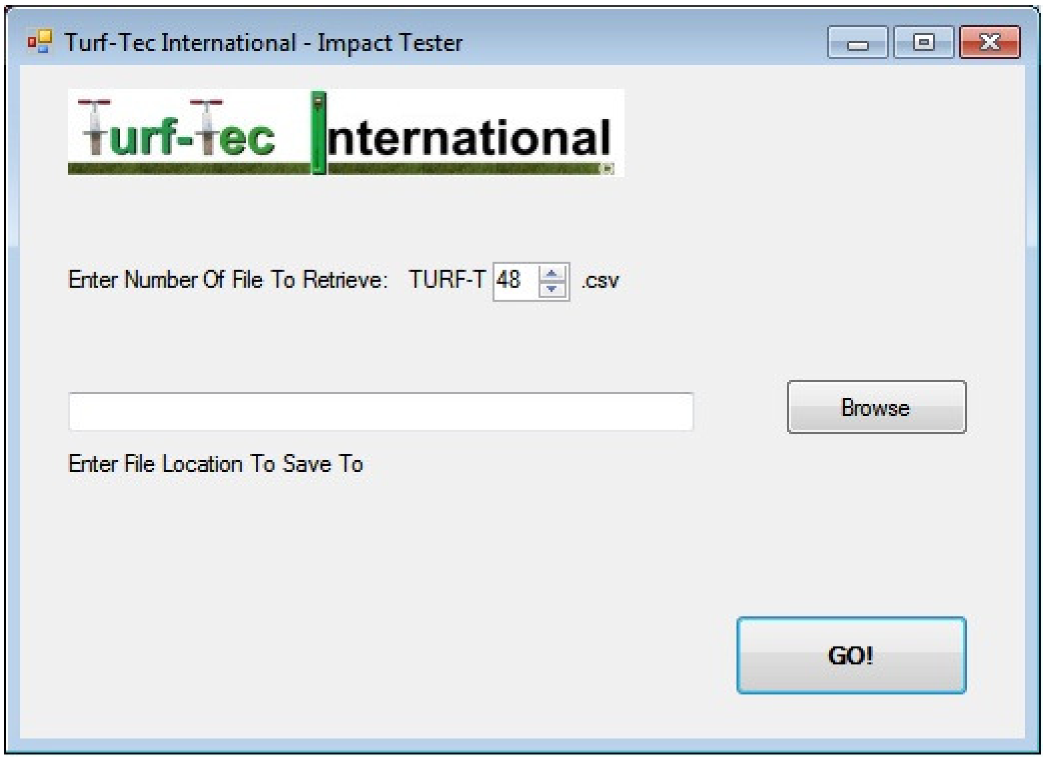
The code that controls communication from the Arduino to the serial port and vice versa has been created and is working properly. This code waits for the file name sent from the PC to the Arduino via serial port and saves it into a character array. The SD card then attempts to open a file using this array. If opening this file is not successful, the Arduino sends an error message back through the serial port. Otherwise, the SD card file is opened and its contents are sent through the serial port to be read and saved by the PC software. This code was created and tested using the Arduino Uno for testing purposes. Once it is completed and tested properly, it will be integrated into the code written for the Arduino MEGA 2650. The code for the serial communication is given below.

```
#include <SD.h>

char myString[12];
char temp;
File myFile;
byte i = 0;
byte j = 0;
// change this to match your SD shield or module;
//     Arduino Ethernet shield: pin 4
//     Adafruit SD shields and modules: pin 10
//     Sparkfun SD shield: pin 8
const int chipSelect = 10;

void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
   while (!Serial) {
    ;// wait for serial port to connect. Needed for Leonardo only
  }


  Serial.print("Initializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by
  // Note that even if it's not used as the CS pin, the hardware S
  // (10 on most Arduino boards, 53 on the Mega) must be left as a
  // or the SD library functions will not work.
   pinMode(SS, OUTPUT);

  if (!SD.begin(chipSelect)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
}


void loop()
{
   if (Serial.available() > 0) {

     if ( i < 12) {
       temp =Serial.read();
      myString[i] = temp;
      i++;
     }

     if (i == 12)  {
        myString[i] ='\0';
       Serial.print("You typed: ");
       Serial.println(myString);
       //i = 0;

            // Test if file exists, if not send an error. If so,
  myFile =SD.open(myString);
  for(j=0; j<13; j++){
     myString[j] ='\0';
    i = 0;
  }

  if (myFile) {

    // read from the file until there's nothing else in it:
    while (myFile.available()) {
         Serial.write(myFile.read());
   }

    // close the file:
    myFile.close();

   }
  else {
         // if the file didn't open, print an error:
    Serial.print(myString);
    Serial.print(" does not exist");
   //Serial.println("X");
  }
     }
   }
}
```

3.4.4 Status of PC Software

We have not been able to test the code yet because of other issues with the software. Once those bugs are fixed we will be able to test the software. We will be putting in print statements in order to see what functions would work or not work.

Below is an example of the Graphical User Interface that was developed.



Turf-Tec International - Impact Tester

Enter Number Of File To Retrieve: TURF-T 48 .csv

Enter File Location To Save To

3.4.5 Status of Alternatives Considered

Characteristic	SD Card
Size	24 x 32 x 2.1mm
Maximum Available Capacity	4GB
Power Requirements	3.3V 20–100 mA
Pins	9
Backup Battery	Yes (in Shield)

3.4.6 Overall Status

The Micro SD card has not been tested because of software problems. When plugged into the Arduino the light on Mirco SD lights up which means the connections are properly placed.

3.5 User Interface

**All technical risks can be seen in the Risk Assessment section of this report.*

3.5.1 Status of User Interface

The interface provides the user with the ability to control the data being saved and acquired from the Impact tester. The user interface is split into three major categories: liquid crystal display (LCD), push buttons, and light-emitting diodes.

3.5.2 Status of Liquid Crystal Display

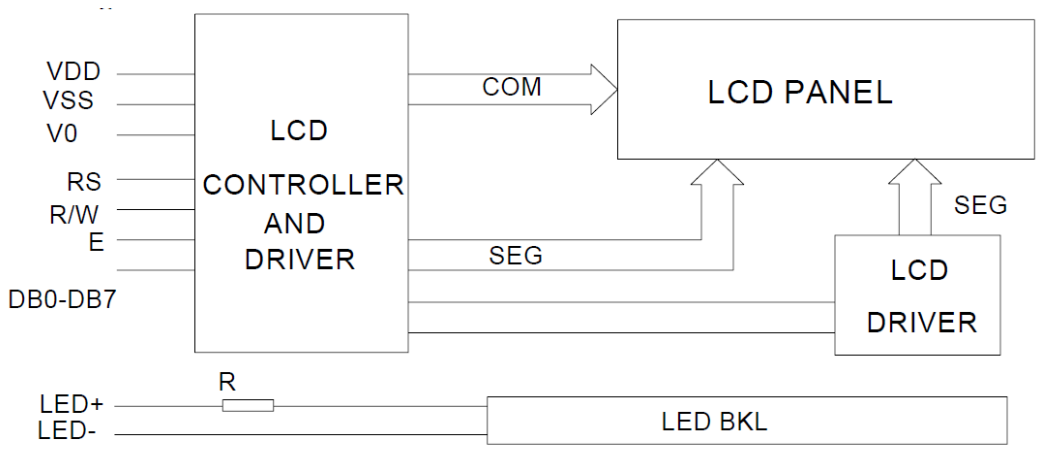


Figure 10: LCD control system

The LCD-00256 is a basic 20 character by 4 line display and utilizes the HD44780 parallel interface chipset. The final product should display the date, time, coordinates, battery level, and GMAX values. The figure below shows the setup of how the LCD is connected to the Arduino MEGA2560.

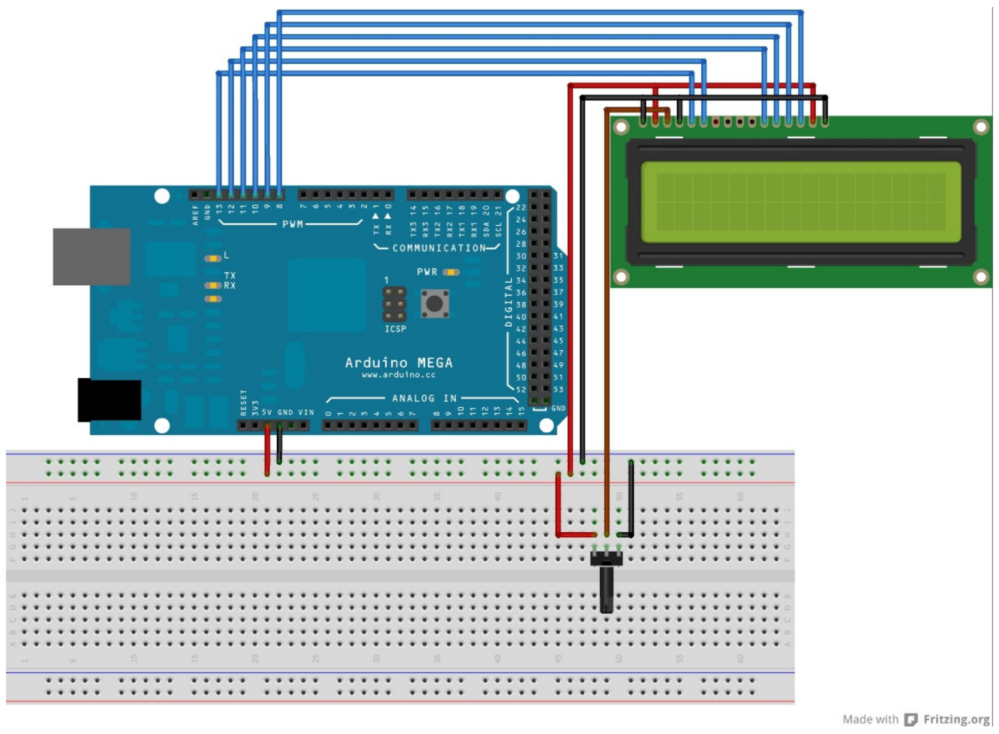


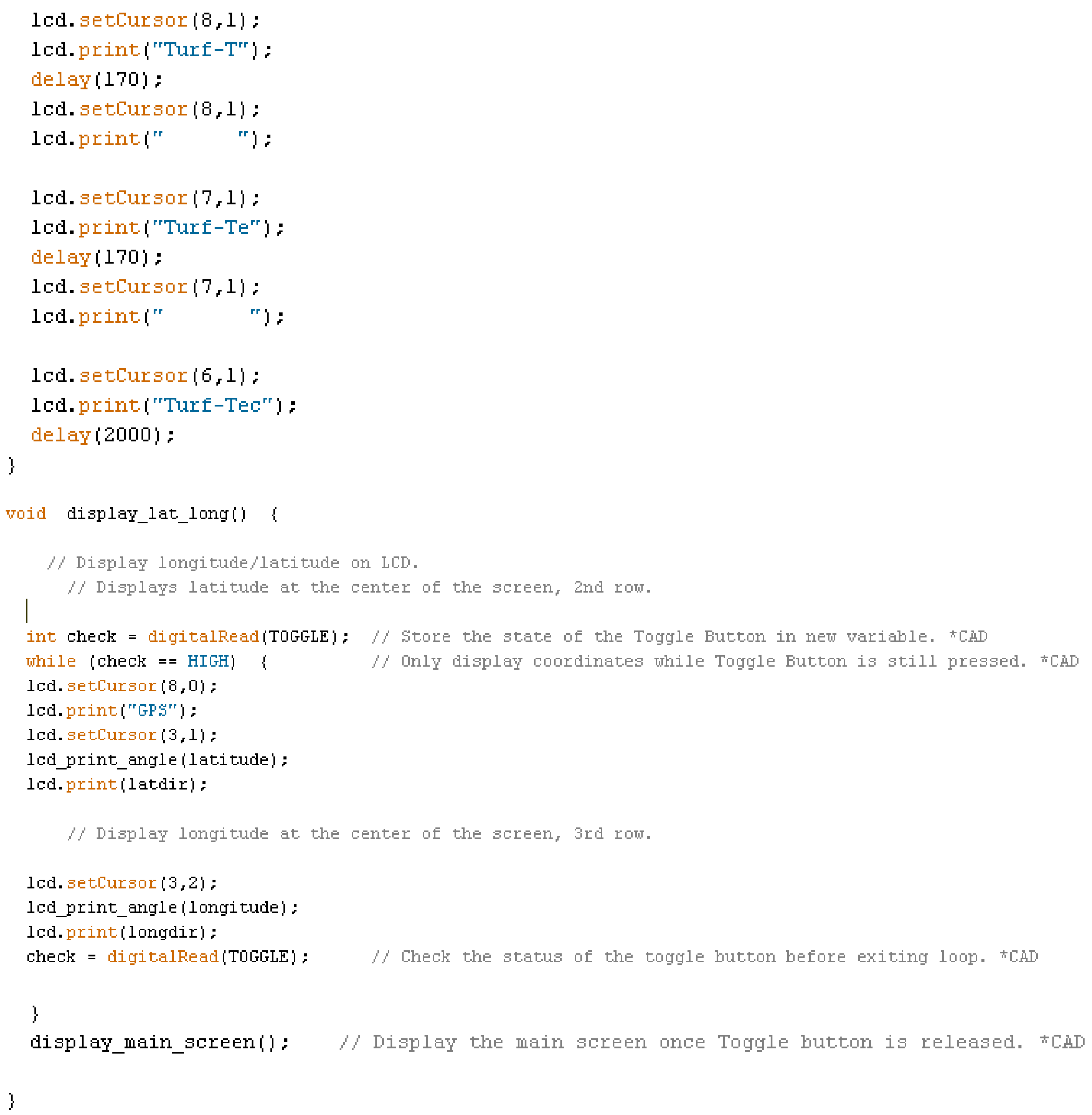
Figure 11: LCD connection to Arduino

The LCD will be controlled by the Arduino board, any information communicated from this controller will be displayed clearly on the screen when powered on. There is a 16 pin connection between the two components. Six of the pins will be connected to the digital pin assignments on the Arduino board, two connected to ground, and one to a 5V output to the Arduino board. The interface pin connection is shown in the table below.

Pin no.	Symbol	External connection	Function
1	V_{SS}	Power supply	Signal ground for LCM (GND)
2	V_{DD}		Power supply for logic (+5V) for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power Supply	Power supply for BKL (Anode)
16	LED-		Power supply for BKL (GND)

The code provided from last year's team works efficiently with the prototype. The LCD displays the correct information properly on the screen. The code provided at the bottom is what the LCD screen shows upon start up and the latitude/longitude display.

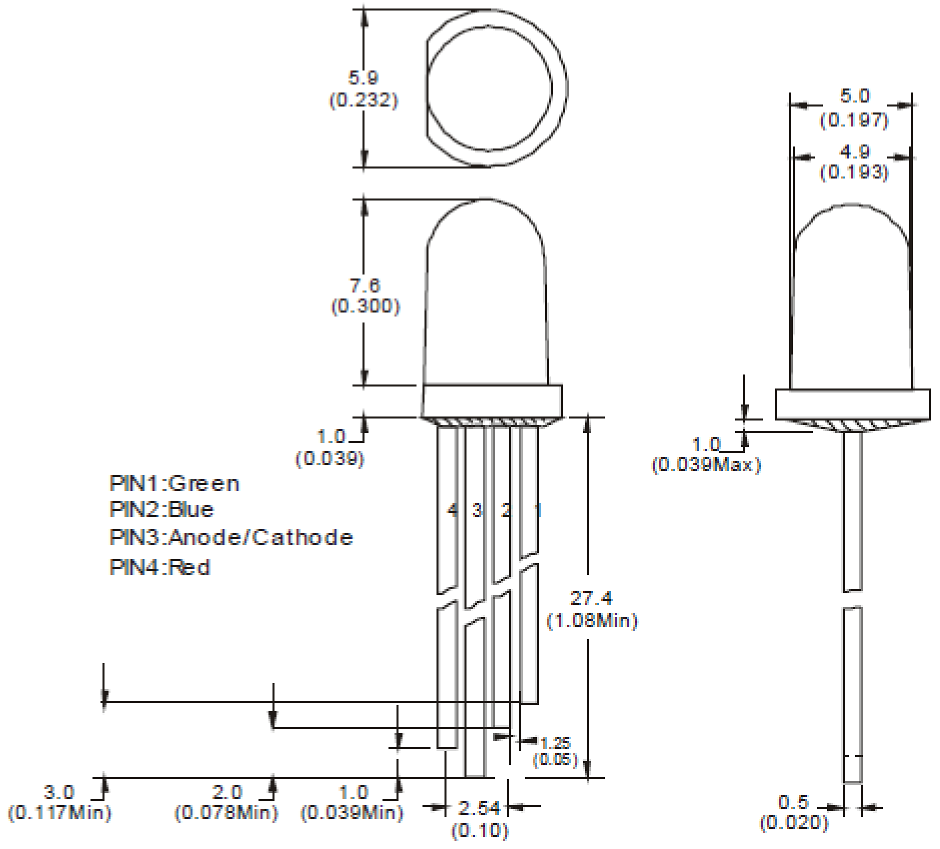
```
            // Scrolls Turf-Tec International at start-up. *CAD
            // Assumes LCD has already been initialized.
            // Used By: [App] setup().

void  disp_logo()  {
  lcd.setCursor(13,1);
  lcd.print("T");
  lcd.setCursor(3,2);
  lcd.print("International");
  delay(170);
  lcd.setCursor(13,1);
  lcd.print(" ");

  lcd.setCursor(12,1);
  lcd.print("Tu");
  delay(170);
  lcd.setCursor(12,1);
  lcd.print("  ");

  lcd.setCursor(11,1);
  lcd.print("Tur");
  delay(170);
  lcd.setCursor(11,1);
  lcd.print("   ");

  lcd.setCursor(10,1);
  lcd.print("Turf");
  delay(170);
  lcd.setCursor(10,1);
  lcd.print("    ");

  lcd.setCursor(9,1);
  lcd.print("Turf-");
  delay(170);
  lcd.setCursor(9,1);
  lcd.print("     ");
```



```
  lcd.setCursor(8,1);
  lcd.print("Turf-T");
  delay(170);
  lcd.setCursor(8,1);
  lcd.print("      ");

  lcd.setCursor(7,1);
  lcd.print("Turf-Te");
  delay(170);
  lcd.setCursor(7,1);
  lcd.print("       ");

  lcd.setCursor(6,1);
  lcd.print("Turf-Tec");
  delay(2000);
}

void  display_lat_long()  {

    // Display longitude/latitude on LCD.
      // Displays latitude at the center of the screen, 2nd row.

  int check = digitalRead(TOGGLE);  // Store the state of the Toggle Button in new variable. *CAD
  while (check == HIGH)  {          // Only display coordinates while Toggle Button is still pressed. *CAD
  lcd.setCursor(8,0);
  lcd.print("GPS");
  lcd.setCursor(3,1);
  lcd_print_angle(latitude);
  lcd.print(latdir);

      // Display longitude at the center of the screen, 3rd row.

  lcd.setCursor(3,2);
  lcd_print_angle(longitude);
  lcd.print(longdir);
  check = digitalRead(TOGGLE);      // Check the status of the toggle button before exiting loop. *CAD

  }
  display_main_screen();    // Display the main screen once Toggle button is released. *CAD

}
```

The first segment simply scrolls “Turf-Tec International” across the screen as soon as the component is powered on. The second segment shows how the screen is now split up, where the GPS coordinates will be displayed on its own separate screen

3.5.3 Status of Light-Emitting Diodes (LEDs)



Block Diagram Specifications of LEDs
Courtesy of Adafruit (BL-L515RGBW-CA Datasheet)

The LED connections are simple, if only one led is being used the cathode is grounded with a resistor in between and the chosen color is connected to an input voltage source. The remaining leg will be connected to a digital output pin of the Arduino board and it will be programmed to either turn on or off on a certain condition. The figure below shows the proper wiring for the tri-color LEDs and the exact pin connections for the Arduino that will be used in the project.

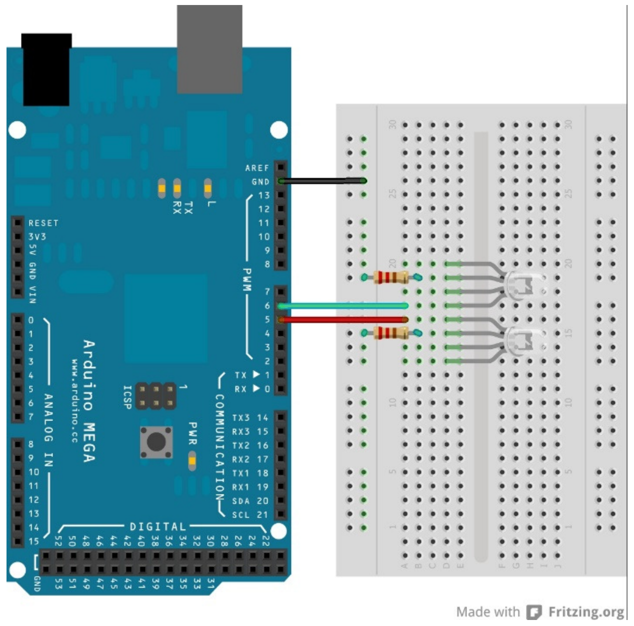


Diagram of tri-color LED connections

When the code was uploaded, the LEDs responded correctly and lit up with their corresponding functions. The main functions for the programming of the LEDs are shown below.

```
    //|==========================================================================================|
    //|    5.  [App] Main application functions.                               [code section]   |
    //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

        //|------------------------------------------------------------------------------|
        //|  setup()  - Main initialization routine of application.  Called on startup.  |
        //|vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv|

void setup()  {

    // Initialize external LEDs.
  init_LEDs();  // Initialize the LED pins.

    // Initialize pushbutton pins.
  init_buttons();  // Initialize the pushbutton pins.

    // Initialize the LCD display.
  init_lcd();
```

The LED pins are initialized as outputs so that the microcontroller can determine when each LED should be turned on.

```
  // Used By: [App] setup().
void  init_LEDs()  {    // Initializes the LED control interface

    // Configure the LED pins as outputs:

//  pinMode(INT_LED_PIN,   OUTPUT);  // Internal LED on pin 13 (not visible to end-user).
  pinMode(GREEN_LED_PIN, OUTPUT);  // Green LED on front of box.
  pinMode(RED_LED_PIN,   OUTPUT);  // Red LED on front of box.

    // Make sure both LEDs are OFF initially.  (They are wired as active-high, so LOW=off.)
    // (A bit later in the startup sequence, we will turn them both ON as a self-test.)

  digitalWrite(GREEN_LED_PIN, LED_OFF);
  digitalWrite(RED_LED_PIN,   LED_OFF);
}
```

The Red LED is used to indicate any errors that may occur and when data is being saved. This can be seen from the following code segments.

```
    // Turn green LED off, red LED on to indicate error.

  digitalWrite(GREEN_LED_PIN, LED_OFF);
  digitalWrite(RED_LED_PIN, LED_ON);
```

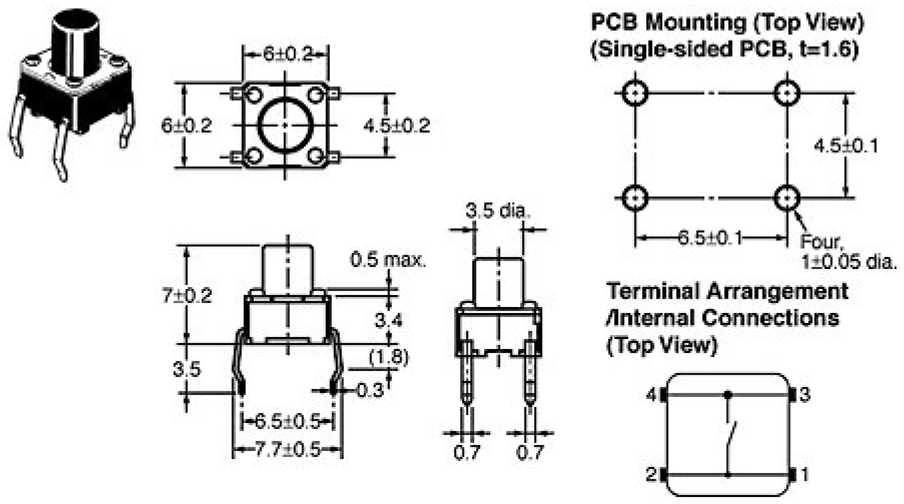
```
void sd_save_data() {

  digitalWrite(RED_LED_PIN, LED_ON);  // turn RED LED on while saving data
  delay(800);
```

The Green LED is used to indicate when data is being acquired. This pertains to incoming GPS coordinates and when a pulse is read from the Accelerometer.

```
digitalWrite(GREEN_LED_PIN, LED_ON);  // Turn on green LED to confirm that a pulse is being / has been detected.
read_pulse();                         // Read the pulse data.
```

```
if (incoming_GPS_data()) {

    // We flash the green LED 'on' just while we are reading serial data.

  digitalWrite(GREEN_LED_PIN, LED_ON);
```

3.5.4 Status of Push Buttons



Block Schematic of Push Buttons
Courtesy of Micros Electronics (6x6mm PB Datasheet)

The team will be using the same push buttons from last year which are 12X12mm which are easily accessible. For example, if maintenance crews were to wear gloves they would have no problem pressing push buttons on the interface. The diagram below shows the push button connect to the Arduino Board.

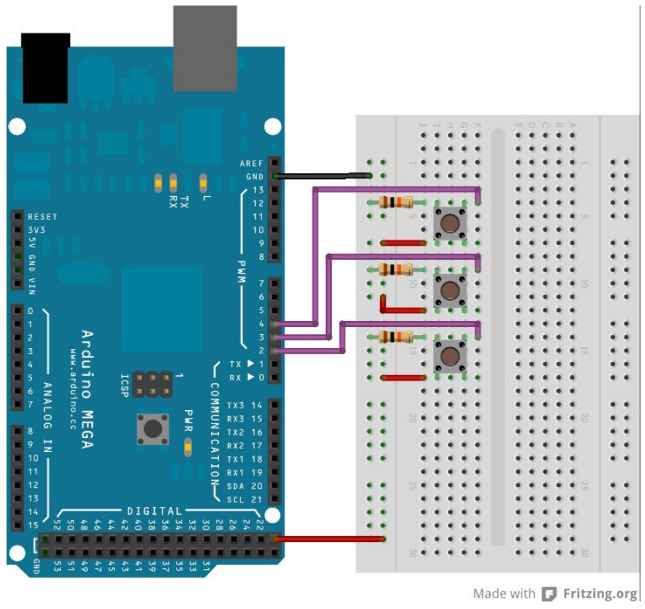
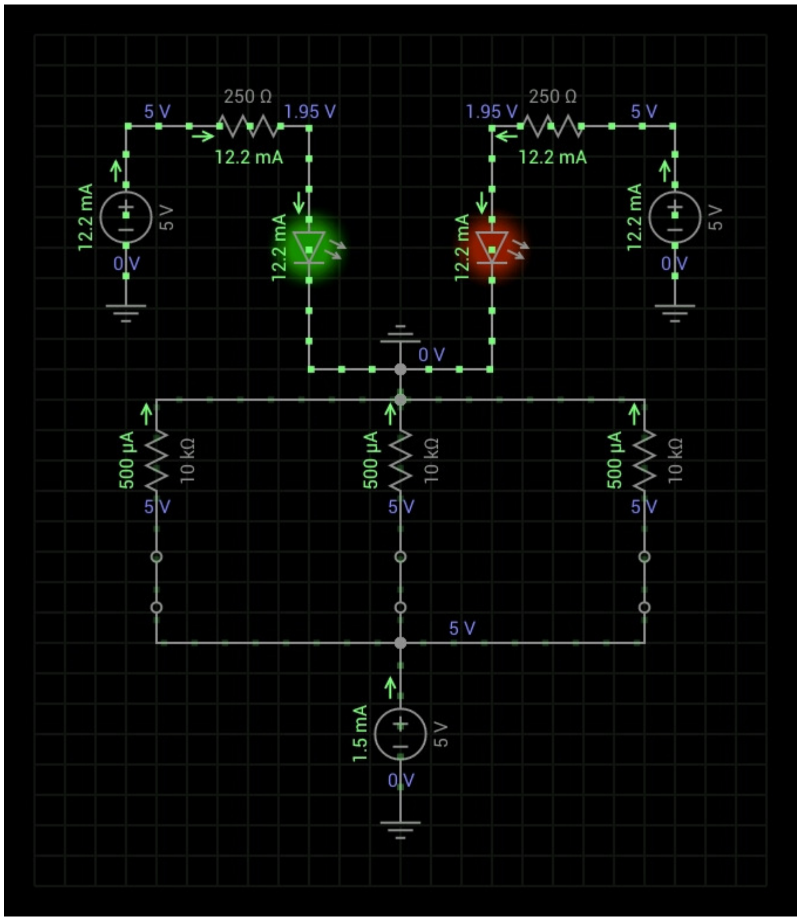


Diagram of push button connections



Push Button LED schematic

The schematic above shows the amount of current being drawn from each component, therefore, 5V sources have been placed before each LED. The sources are jumped to the Arduino digital pins 6 (Green LED) and 5 (Red LED). There is a total of four push buttons, one which is the power button, each of these buttons have their own functionality. The first push button is the save button, the second is the clear button which clears the screen of current GMAX value, and the last button is the toggle button which switches in between the main screen and GPS screen. The connections of each push button are simple; 5V will be used as the input voltage and a 10k resistor will be connected between the push button and ground. The large resistor value will cause less current to be drawn (total of 1.5mA) and will reduce the power consumption of this board. The leg of each push button connected to the 5V will also be connected to a digital pin of the Arduino board which will provide the programming behind the status of each switch. The save button will be connected to Arduino pin 4, the clear button to Arduino pin 3, and the toggle button to Arduino pin 2. In the main loop of the code, the buttons are constantly being checked to see if any have been pressed. This is done by the function check_buttons(). When this function is called, each button is checked individually.

```
void  check_buttons()  {
  check_save_btn();
  check_disc_btn();
  check_togg_btn();      // New toggle button has been added. *CAD
}
```

When the save button is checked, it must also check if the clear button has been pressed. If both are true, then the screen will display the time zone settings. If only the save button has been pressed, its function will be processed.

```
void  check_save_btn()  {

  unsigned long now_millis = millis();

    // Process expiry of refractory period for save button.

  if (save_refractory)  {
    unsigned long elapsed = now_millis - last_save_millis;
    if (elapsed >= PUSHBUTTON_DEBOUNCE_MILLIS) {
      save_refractory = false;
    }
    else {
      return;  // Still in refractory period, so don't even bother reading button.
    }
  }

    // Get the current state of the "save" button.

  int  new_state  =  digitalRead(SAVE_BUTTON_PIN);

    // If the state is unchanged, there's nothing else for us to do here.

  if (new_state == save_ButtonState) return;

    // Remember the new state.

  save_ButtonState = new_state;

  save_ButtonState = new_state;

    // If save button was just pressed, process that event.

  if (save_ButtonState == BUTTON_STATE_PRESSED) {
    last_save_millis = now_millis;
    process_saveButtonPress();
  }
}
```

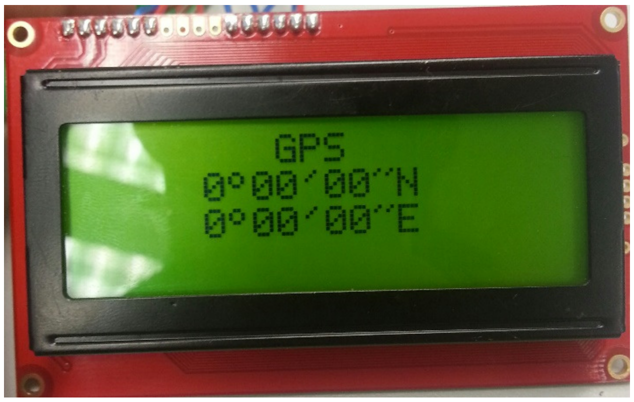
When the clear button is checked, it must also check if the save button has been pressed. If both are true, then the screen will display the time zone settings. If only the clear button has been pressed, its function will be processed.

```
void  check_disc_btn()  {

  unsigned long now_millis = millis();

    // Process expiry of refractory period for save button.

  if (disc_refractory)  {
    unsigned long elapsed = now_millis - last_disc_millis;
    if (elapsed >= PUSHBUTTON_DEBOUNCE_MILLIS) {
      disc_refractory = false;
    }
    else {
      return;  // Still in refractory period, so don't even bother reading button.
    }
  }

    // Get the current state of the "discard" button.

  int  new_state  =  digitalRead(DISCARD_BUTTON_PIN);

    // If the state is unchanged, there's nothing else for us to do here.

  if (new_state == discard_ButtonState) return;

    // Remember the new state.

  discard_ButtonState = new_state;


    // If discard button was just pressed, process that event.

  if (discard_ButtonState == BUTTON_STATE_PRESSED) {
    last_disc_millis = now_millis;
    process_discButtonPress();
  }

}
```

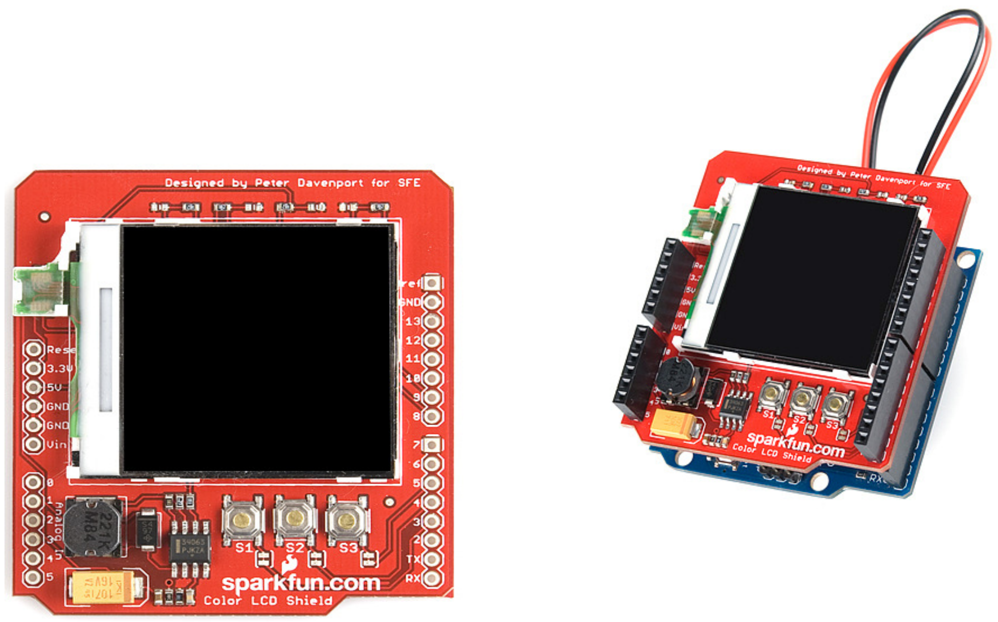
When the toggle button is pressed, it will clear the screen and only display the GPS coordinates. The following code segments will show the main functions required for this feature and the final process of the push button.

```
void  check_togg_btn()  {

  toggle_ButtonState  =  digitalRead(TOGGLE);

  if (toggle_ButtonState == BUTTON_STATE_PRESSED) {
    process_toggButtonPress();
  }
}

void process_toggButtonPress()  {      // This function will process once the Toggle button is pressed. *CAD

#ifdef SERIAL_DIAGNOSTICS
  Serial.println(F("TOGGLE pushbutton pressed."));
#endif

  digitalWrite(GREEN_LED_PIN, LED_OFF);

  lcd.clear();                         // Clear the screen before displaying the GPS coordinates. *CAD
  display_lat_long();                  // Display GPS coordinates.

} // end process_toggButtonPress().
```

The picture below shows an example of the toggle screen.



3.5.5 Status of Alternative Consideration



Color LCD shield
Courtesy of Sparkfun electronics

The price difference is not much greater and it satisfies majority of the requirements needed for the component. The board includes three push buttons, a white LED backlight, uses the Epson S1D15G10 or Philips PCF8833, and is accessed through 9 pins. The color display is not needed and the dimensions (1.2x 1.2) proved to be too small for the amount of information needed to be displayed. Many programming changes would need to be made in order to format the data displayed properly and have a clean outlook.

3.5.6 Overall Status

The Liquid Crystal Display, push buttons, and LEDs have already been purchased from last year's team. So far we have tested the LCD screen and it has been displaying blocks so the team is testing the screen to see where and why this error is occurring.

3.6 Power System (Battery)

**All technical risks can be seen in the Risk Assessment section of this report.*

3.6.1 Status of Power System

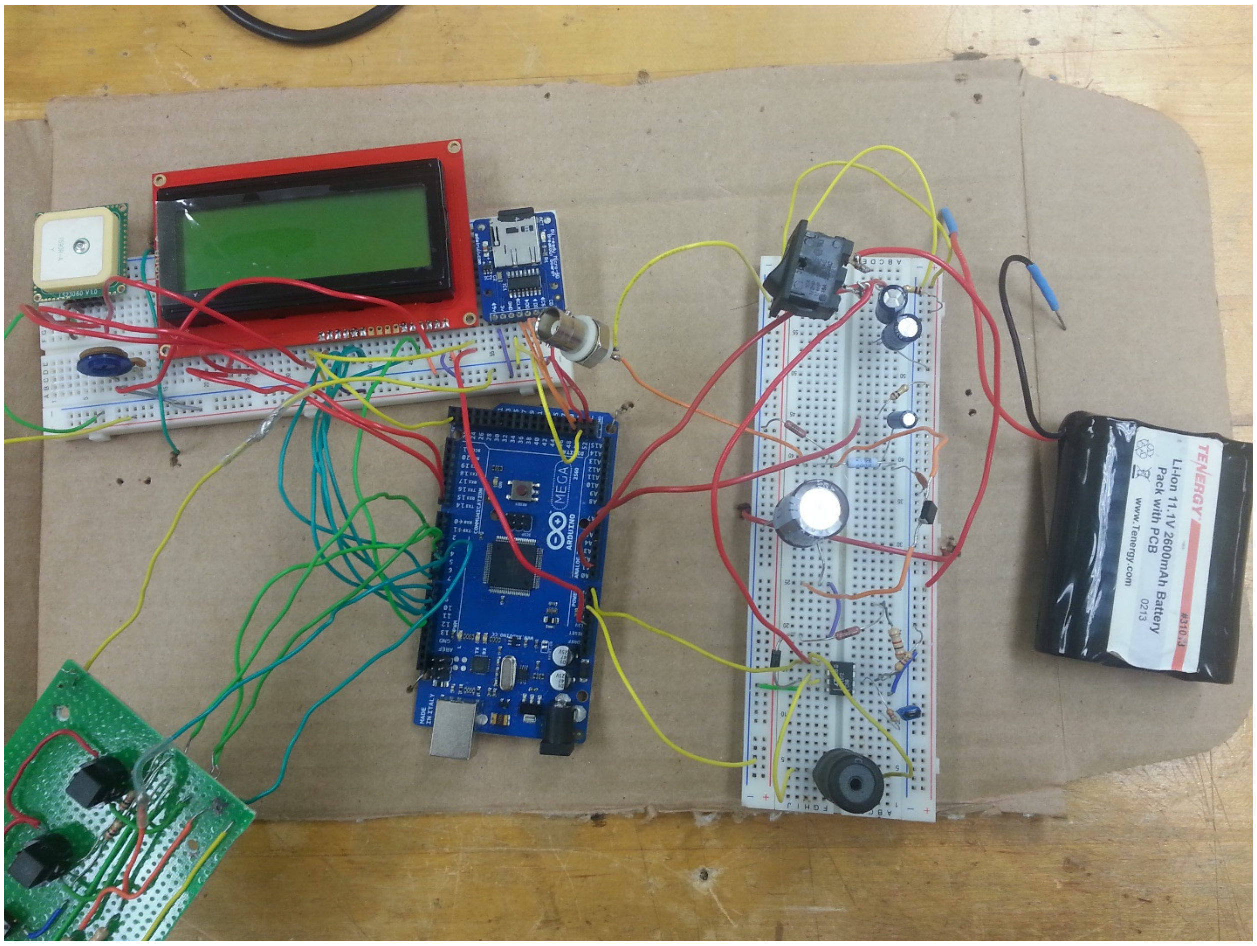
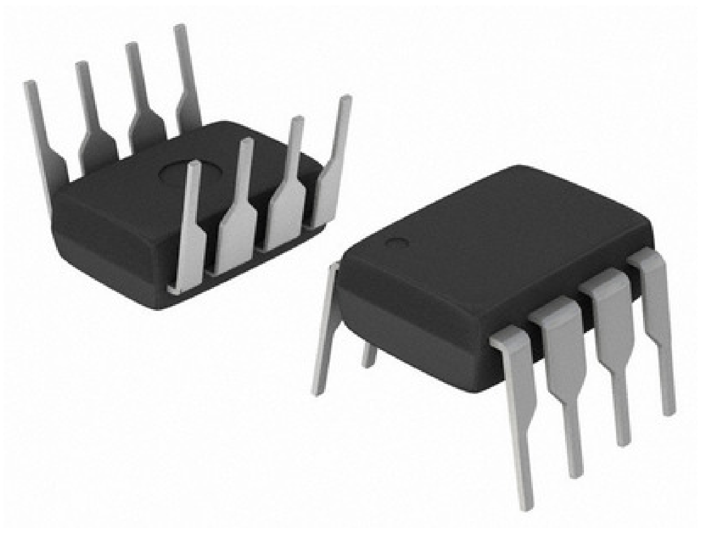


Figure 12: shows the boost converter

The battery source uses an 11.1V voltage source with a power switch in between the capacitor that is in parallel with the battery and Pololu (which is our boost converter). The 11.1 V will be directly connected to the Arduino; this is because for one, the Arduino can take up to 7-12 V and that it has a built in fixed voltage regulator that drops the voltage to 5 V(which will going to the GPS, Micro SD card, LCD, pushbuttons and LEDs). The 11.1 V then goes to the Pololu; From the Pololu, the voltage is then boosted up to 24.5 V by using the potentiometer located on the Pololu. The circuit includes a current regulator which helps limit the amount of current going into the potentiometer. Last year's team found a suitable resistor by using ohm's law. The equation below shows how the resistance was calculated.

$$\boldsymbol{R} = \frac{V_{ref}}{I} = \frac{1.25\ V}{13\ mA} = 96 \approx \mathbf{100}\ \Omega$$

After finding the resistor value it was then placed in series between the output and adjust pin. V_{ref} is the reference voltage at the adjust pin and I is the current. A load resistance of 1.8kΩ to cut the accelerometer current down to 4 mA. The bandpass filter is used to filter out the high frequency glitches and also used as a DC blocker to let the signal pass to the Arduino for the calculations of the GMAX values.



LT1072 chip from digikey.com

A high efficient chip (LT1072) shown above will verify a fixed voltage that the accelerometer can operate in and save some labor that Mr. Mascaro wouldn't have to worry about. With this chip last year's team was able to boost the voltage up to a fixed voltage of 40.9 V.They also reintroduced the constant current diode model (CLD20B) which has high voltage intake and limits the current to 20mA. To be able to boost the voltage we had to identify the values for the inductor,capacitor and resistors. Last year's team found the inductor and capacitance values; doing some research and having prior knowledge in power electronics they configured the inductance and capacitance values by using the following equations:

$$\boldsymbol{L} = \frac{V_g D T_s}{2\Delta i_L} = \boldsymbol{60\mu H}$$

And

$$\boldsymbol{C} = \frac{\Delta i_L T_s}{8\Delta v} = \boldsymbol{200\mu F}$$

Where V_g is the input voltage, D is the duty cycle, T_s is the switching period, Δi_L is the current ripple of the inductor and Δv is the voltage ripple of the capacitor.

To identify the resistor values we used:

$$Vout = 1.3\left(1 + \frac{R_1}{R_2}\right)$$

Where 1.3 V is the reference voltage between feedback and ground. Setting R_2 = 470Ω wanting an output voltage between 30-40 V; solving for R_1:

$$\boldsymbol{R_1} = \left(\frac{40V}{1.3V} - 1\right)470\Omega = \boldsymbol{13k}\Omega$$

With the resistance values calculated we were able to have an output voltage of 40.9V; plus with this resistor setup we were able to control the stability of the voltage due to the feedback being connected in between the resistor values. The purpose of output voltage being a higher was because we have implemented the constant current diode which has a voltage drop(in our case 21.4 V) then the load resistance(1kΩ) cuts the current in half and has a voltage drop of 19.65 V which is suitable for the accelerometer. Table 3-1 shows the power consumption of the main components in our system.

Table 3-1

	Voltage (V)	Current (mA)	Power Consumption (W)
Accelerometer	19.65	10	196.5m
Arduino	11.1	320	3.552
CLD20B	21.4	20	428m
GPS	3.3		
LT1072	Voltage in: 11.1	Current in: 80	Power in:884.6m
	Voltage out: 40.9	Current out: 20	Power out:813.85m
SD card	3.3	100	330m

3.6.2 Status of Battery

The Tenergy Li-Ion 11.1V is plugged directly into the Arduino and then the internal voltage regulator converts the 11.1V into 5V, which powers the GPS, push buttons, LEDs, LCD, and Micro SD. The battery has a boost chip (model LT1072) which ups the voltage to an operating range of (18-30V) for the accelerometer. The picture below shows that that battery and boost are applying adequate amount of voltage to the accelerometer.

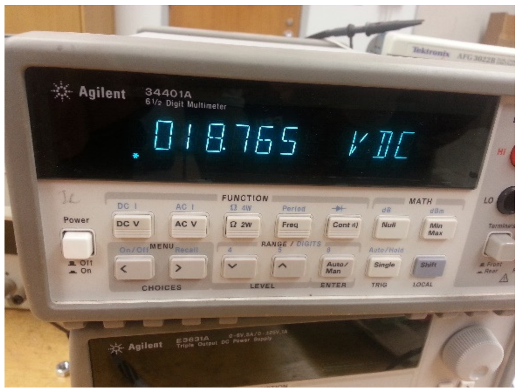


Figure 20: Accelerometer operating voltage and current battery being used
(Tenergy Li-Ion 11.1V 2600mAh)

The battery is charged using a Universal Smart battery charger which is designed to charge 11.1V Li-Ion/Polymer battery with a capacity greater than 2200mAh.

1.8A for 11.1V Li-Ion/Polymer Charger from batteryspace.com

Battery charger specification:

- Voltage of 100-240VAC
- Current of .3VAC
- Max power at 40W and rating frequency to be 50/60Hz
- Output of 12.6VDC at 1.8A
- Automatic cutoff at 12.6V and output shorted and overload protection
- 5.5 x 2.1 x 10mm male barrel plug

3.6.3 Status of Alternatives Considered

Characteristic	NiMH Battery w/ Bare Leads	Li-Ion 18650 Battery	Tenergy Li-Ion 18650
Website	All-battery.com	Batteryspace.com	All-battery.com
Voltage	24V	25.9V	25.9V
Current	2000mAh	2600mAh	2600mAh
Dimensions	141mm x 50mm x 8mm	135mm x 35mm x 74mm	73mm x 68mm x 40mm
Weight (oz)	18.34	14.4	12.7

Battery Comparison Chart courtesy of All-Battery.com

3.7 Printed Circuit Board

**All technical risks can be seen in the Risk Assessment section of this report.*

3.7.1 Status of Printed Circuit Board

The design process of the Printed Circuit board is currently in its early stage. The schematic and board interface have been changed completely, simplifying the circuit board design. The team decided on placing all the necessary components in one single board with dimension of 5.2” x 4” x 0.1” to match the impact tester main box. The single board will include the LCD, pushbutton, LEDs and the external power supply. The capacitor connection and the components connected to it are in a questionable state until a decision is made on what type of capacitor we’ll have (whether through hole or surface mount) to reduce the circuit board width.

3.7.2 Status of Software Selected

As stated in the previous report, the software chosen to design the PCB was Eagle CAD and some SketchUp to generated 3D visualization. A few group members have downloaded the program onto their personal computers and have started learning the program. We took the previous design from last year and modified it for our new design.

3.7.3 Status of Interface Layers

The interface is composed of two layers of connections. The top layer contains all the components as well as most of the red traces while the bottom layer only contains blue traces. The traces on the top layer are connected to the bottom layer via the yellow traces.

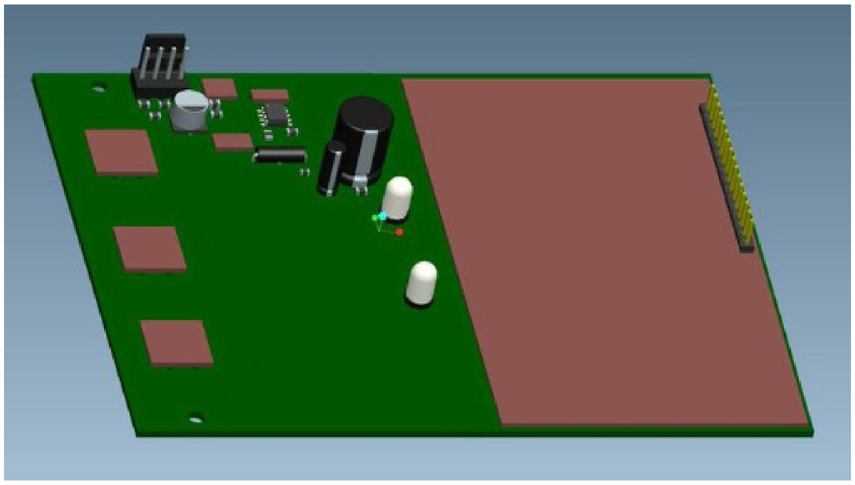


Figure 3.7.01 Interface Board in 3D

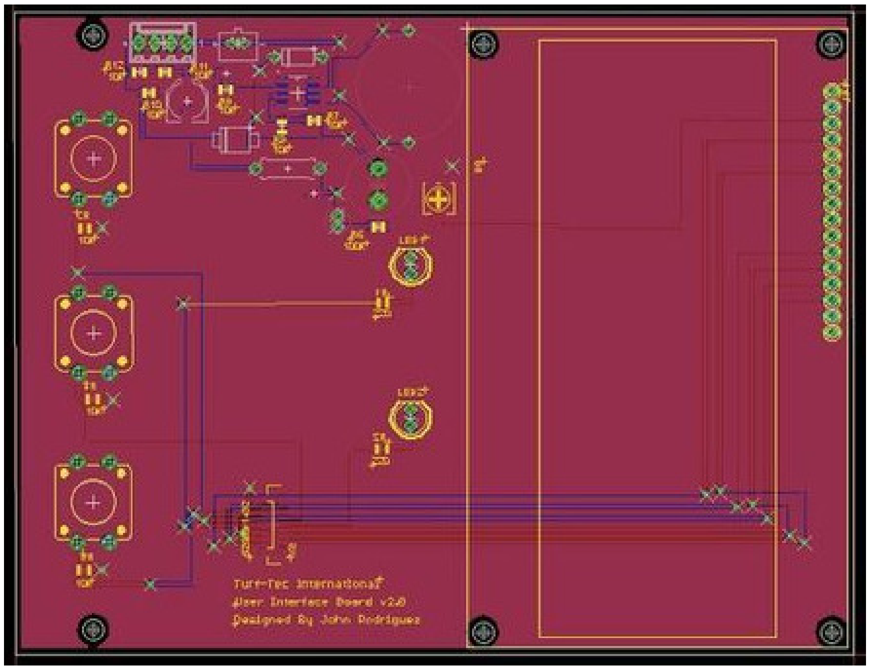


Figure 3.7.02 Interface Top Layer

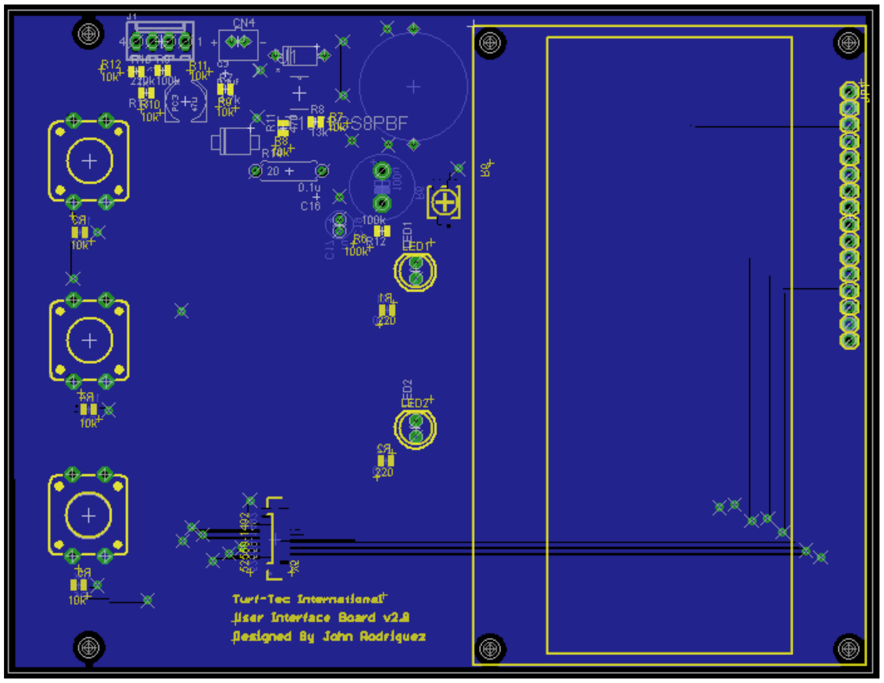
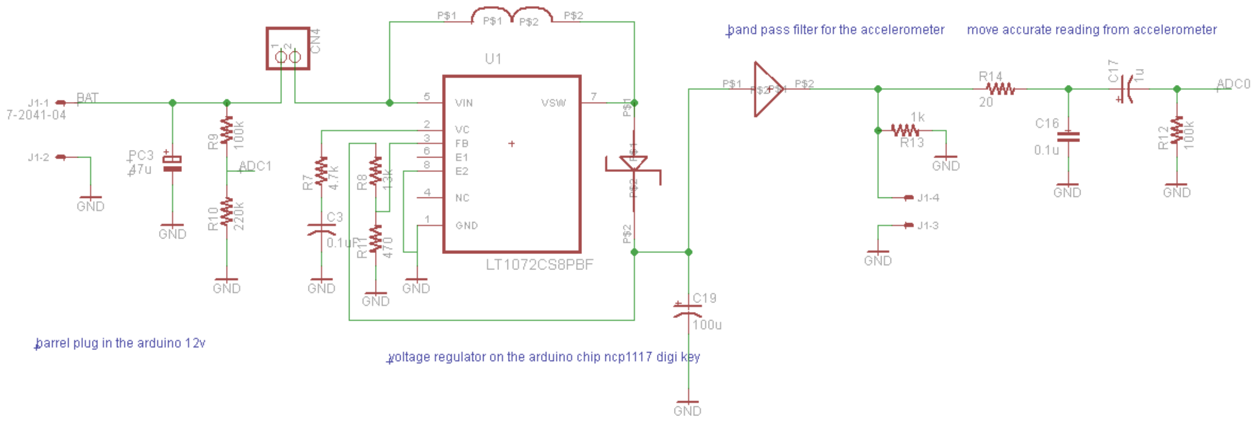


Figure 3.7.03 Interface Bottom Layer


BAT
7-2041-04
CN4
U1
LT1072CS8PBF
VIN
VSW
VC
FB
E1
E2
NC
GND
R9 100k
R10 220k
ADC1
PC3 47u
R7 4.7k
R8 13k
R11 470
C3 0.1uF
C19 100u
band pass filter for the accelerometer
move accurate reading from accelerometer
R13 1k
R14 20
C16 0.1u
C17 1u
R12 100k
ADC0
J1-4
J1-3
barrel plug in the arduino 12v
voltage regulator on the arduino chip ncp1117 digi key
Figure 3.7.04 Voltage Booster Convertor

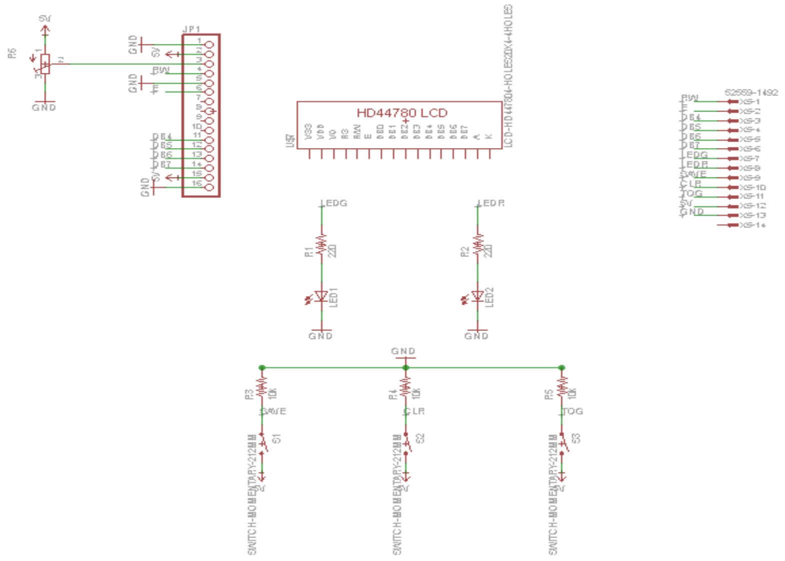


Figure 3.7.05 LCD & LEDs schematics.

3.7.4 Status of Unnecessary Components

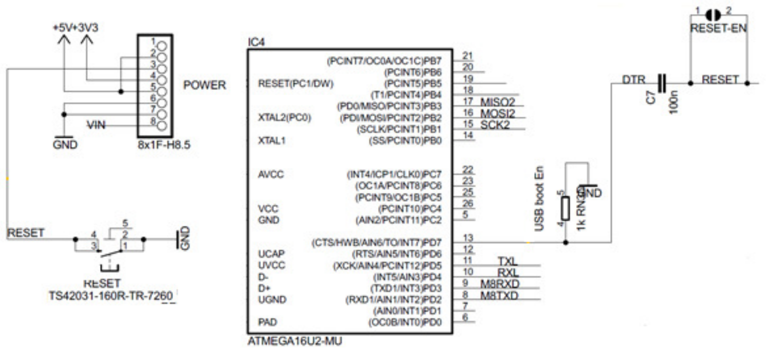


Figure 3.7.06 Reset Schematic

3.7.5 Status of Alternatives Considered

Characteristic	Sketch-Up	POV Ray

Schematic File Extension	.skp	.inc
Board File Extension	Use software for conversions	.pov
3D Viewing	Yes	Yes
Library	Moderate	Extensive
Ease of Use	Hard	Very Hard

3.7.6 Overall Status

Currently in the design process, we have begun creating a board with the new layout. The Eagle CAD software is being applied to do this using the Gerber files. This process is maintaining the learning experience and further familiarizing the team in the new software.

4 Schedule

Task	**Start Date**	**Duration (Days)**	**End Date**	**Assigned Team Members**
Learn CAD Software	10/29/13	37	12/14/13	Shaneetra Graham John Rodriguez
Ordering Parts	11/20/13	10	11/30/13	Joyce Kosivi
Testing of Individual Modules	9/26/13	53	12/7/13	All members
Implement Arduino Board	10/29/13	14	11/15/13	John Rodriguez
Design PCB Board	11/26/13	31	1/7/13	All members
Testing GPS code	10/29/13	33	12/10/13	Christian Rodriguez
Design 3D models in AutoCAD	10/29/13	37	12/14/13	John Rodriguez
Updating Website with new code and changes	10/29/13	33	12/10/13	John Rodriguez
Finding out errors in code	10/29/13	33	12/10/13	Christian Rodriguez Shaneetra Graham
Soldering of board	1/7/14	10	1/17/14	Johnnie McCormick
GPS initialization code working	1/12/14	4	1/16/14	Christian Rodriguez

A list of the areas of code that has been changed and why	1/12/14	5	1/17/14	Shaneetra Graham Christian Rodriguez
Pictures of your PCB schematic so far	1/12/14	5	1/17/14	Joyce Kosivi
Pictures of user interface	1/14/14	3	1/17/14	Shaneetra Graham
PCB sketch review	1/17/14	4	1/22/14	John Rodriguez
Milestone #4	1/28/14	6	2/3/14	Everyone

5 Budget Estimate

A. Personnel	Total Hours	Base Pay	Total
Computer			
Christian Rodriguez	396	$30.00	$11,880.00
Shaneetra Graham	396	$30.00	$11,880.00
Electrical			
Joyce Kosivi	396	$30.00	$11,880.00
John Rodriguez	396	$30.00	$11,880.00
Johnny McCormick	396	$30.00	$11,880.00
Subtotal of A.			$59400.00
B. Fringe Benefits		25% of A.	$14850.00
C. Personal overhead cost		45% of A.	$26730.00
D. Total Personnel Costs			$100,980.00

E. Expenses

Item	Cost	Quantity	Shipping	Total
LCD	Purchased	1	$0.00	$17.95
GPS	Purchased	1	$0.00	$59.95
Arduino MEGA 2560	Purchased	1	$0.00	$49.95
Accelerometer	Purchased	1	$0.00	$299.99

Tenergy Li-Ion Battery	Purchased	1	$0.00	$79.99
MicroSD Shield	Purchased	1	$0.00	$24.95
F. Total Direct Costs				$514.83
G. Overhead Costs			45% of F	$231.67
Total Cost of Project	D + F + G			$101,726.50

6 Overall Risk Assessment

6.1 Technical Risks

Technical risks are design, integration and project completion risks that may impact the success of the project. **Technical risk is expected** in any new design. To not have any technical risk is to basically copy what has already been done. Technical risks can include: new or innovative designs that do not have a certainty for success, new technologies being used that are not completely understood, problems with current designs that must be overcome, solutions to design problems that have not been identified, etc.

6.1.1 Technical Risk: Arduino MEGA2560

6.1.1.1 Technical Risk 1: The Ardruino overheats

<u>Description</u>

If there is too much power going to the Arduino then it will automatically go into standby mode until it cools down

<u>Probability: < Very Low, Low, Moderate, High, or Very High></u>

Moderate: The board can operate on an external supply of 6 to 20 volts. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. The battery provides 11.1V to the Arduino board which falls in the recommended range and the 5V and 3.3V pins are used as output pins. The Arduino has 54 digital pins each usable as input or output pins and each operate at 5V.

<u>Consequences: <Minor, Moderate, Severe or Catastrophic></u>

MODERATE: This will cause the Arduino to not be able to be used for a long period of time because then it will go into standby mode. This is only a moderate risk because the board will still be functional just for a short period of time.

<u>Strategy</u>

1. First, total current will be calculated per pin.

2. If too much current is calculated, steps to reduce power through the board will be done (for example, providing a voltage divider outside of the Arduino so less voltage is used).

6.1.1.2 Technical Risk 2: There is not enough memory

Description

One of the problems the previous years' team had with their previous board (the UNO) was that there wasn't enough memory in the board to support all the code.

Probability: <Very Low, Low, Moderate, High, or Very High>

VERY LOW: There should be more than enough memory to support all the functions of the board

Consequences: <Minor, Moderate, Severe or Catastrophic>

MINOR: If anything code could be reduced or the Micro SD card could be implemented as additional memory

Strategy

1. Write code and check the size of the files needing to load
2. If the file size exceeds 256K, implement the Micro SD card

6.1.2 Technical Risk: GPS

6.1.2.1 Technical Risk 1: GPS isn't accurate

Description

One of the requirements is that the GPS calculates location accurately to avoid repeating tests locations.

Probability: < Very Low, Low, Moderate, High, or Very High>

LOW: The LS20031 has 66 channels to provide precise measurements and differential GPS.

Consequences: <Minor, Moderate, Severe or Catastrophic>

SEVERE: If the GPS is not accurate it is pointless to have it. It has to provide precise location so the user does not repeat a test location.

Strategy

1. Enable the backup battery to make sure that the satellites are located at all times
2. Enable the differential GPS option
3. If GPS still isn't accurate an external DGPS can be implemented

6.1.2.2 Technical Risk 2: Satellites cannot be located

Description

The GPS will not be able to locate satellites because of its location

<u>Probability: < Very Low, Low, Moderate, High, or Very High></u>

VERY LOW: The GPS can provide information in almost any environment. This should not be an issue because the impact tester will not be used in areas that might cause a problem locating satellites.

<u>Consequences: <Minor, Moderate, Severe or Catastrophic></u>

MINOR: The areas that will have difficulty finding satellites will most likely not be the areas being tested due to the intended area of use.

<u>Strategy</u>

1. Provide knowledge as to how a GPS works and explain that this is a low risk with a small consequence.
2. Alternate forms of GPS could be used if location can't be found. Even so, GMAX readings can still be conducted.

6.1.2.3 Technical Risk 3: Backup battery gives out

<u>Description</u>

The backup battery is used to keep satellite locations active at all times

<u>Probability: < Very Low, Low, Moderate, High, or Very High></u>

HIGH: After a certain amount of time the backup battery will eventually die.

<u>Consequences: <Minor, Moderate, Severe or Catastrophic></u>

SEVERE: This consequence proves to be severe only because there will be a long start up time prior to any testing in order for the GPS to acquire satellite locations. This won't stop the function of the Impact Tester itself, however, it would prove as a nuisance to the consumer.

<u>Strategy</u>

1. Provide to the consumer the risk of this happening and explain what they should expect to happen if it does
2. Have a supply of the micro-battery to sell separately
3. Offer to change the battery itself (clip on battery)
4. Allow the consumer to change the battery themselves

6.1.2.4 Technical Risk 4: GPS error correction won't enable

<u>Description</u>

The GPS error correction implemented does not work

<u>Probability: < Very Low, Low, Moderate, High, or Very High></u>

MODERATE: There is a chance that the GPS hardware could be faulty and not allow for the correction to work. If it's not a hardware issue then programming troubleshooting will have to be looked into.

Consequences: <Minor, Moderate, Severe or Catastrophic>

MODERATE: If the GPS error correction doesn't work, there can be a larger error in accuracy than originally anticipate… which will then attribute back to the first technical risk.

Strategy

1. Do research on how to enable the GPS correction
2. Contact the store if it doesn't enable to see if it's a product issue or code issue
3. It will be known if it's a code issue in the NMEA sentence

6.1.3 Technical Risk: Accelerometer

6.1.3.1 Technical Risk 1: Software does not interface with the accelerometer

Description

The software does not convert the electrical pulse to GMAX values.

Probability: < Very Low, Low, Moderate, High, or Very High>

Very low: We will be using the existing code from last year's team to convert the voltage signals to GMAX values.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Severe: The reason why this risk is severe is because the accelerometer plays a large role in this project and if the voltage signals can't get converted then the accelerometer will not serve its main objective which is to measure GMAX values.

Strategy

1. Continue to work with the exiting code and to improve upon it.

6.1.3.2 Technical Risk 2: ICP Malfunction

Description

The vibration or shock does not convert to an electrical pulse.

Probability: < Very Low, Low, Moderate, High, or Very High>

Moderate: The reason why this is moderate is because the accelerometer needs a specific voltage and current drawn into it and if doesn't get it; it won't be operational.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Catastrophic: The reason why this is catastrophic is because without an accelerometer the project wouldn't have any meaning.

Strategy

1. To check the voltage and current in the constructed circuit before connecting the accelerometer.
2. Keep a spare just in case of any mishaps.
3. Turn the power off after every usage.

6.1.3 Technical Risk: Accelerometer

6.1.3.1 Technical Risk 1: Software does not interface with the accelerometer

Description

The software does not convert the electrical pulse to GMAX values.

Probability: < Very Low, Low, Moderate, High, or Very High>

Very low: We will be using the existing code from last year's team to convert the voltage signals to GMAX values.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Severe: The reason why this risk is severe is because the accelerometer plays a large role in this project and if the voltage signals can't get converted then the accelerometer will not serve its main objective which is to measure GMAX values.

Strategy

1. Continue to work with the exiting code and to improve upon it.

6.1.3.2 Technical Risk 2: ICP Malfunction

Description

The vibration or shock does not convert to an electrical pulse.

Probability: < Very Low, Low, Moderate, High, or Very High>

Moderate: The reason why this is moderate is because the accelerometer needs a specific voltage and current drawn into it and if doesn't get it; it won't be operational.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Catastrophic: The reason why this is catastrophic is because without an accelerometer the project wouldn't have any meaning.

Strategy

1. To check the voltage and current in the constructed circuit before connecting the accelerometer.
2. Keep a spare just in case of any mishaps.
3. Turn the power off after every usage.

6.1.4 Technical Risk: Data Logger

6.1.4.1 Technical Risk 1: The USB can't retrieve information

Description

Data cannot be retrieve from the micro SD card.

Probability: < Very Low, Low, Moderate, High, or Very High>

Moderate: code has been written but has not been tested by this year's group yet to make sure it's working correctly

Consequences: <Minor, Moderate, Severe or Catastrophic>

MINOR: If the USB can't retrieve information the old technique or simply pulling a data card out of the Adafruit shield will be implemented.

Strategy

1. Testing needs to be done to make sure the existing code works correctly and if not, then code needs to be modify to make sure it does work correctly.

6.1.5 Technical Risk: User Interface

6.1.5.1 Technical Risk 1: LCD does not display correct information

Description

The LCD should be able to display the correct date, time, coordinates, battery level, and GMAX values.

Probability: < Very Low, Low, Moderate, High, or Very High>

Low: Last year's Turf Tec team developed code that properly interfaces with the same exact LCD model. The code displays all of the correct information after each trial run.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Catastrophic: If the LCD screen does not display the proper information, the entire Impact Tester is defected. The screen is the heart of the interface which directly shows the user exactly what is being read and measured. If interface was defected, the user would lose complete control of the product.

Strategy

1. Verify that the data logger is storing the proper information. If this is also wrong, the problem could be elsewhere.
2. Check that the connections are jumped correctly to each pin of the Ardiuno Board.
3. Debug the code that is communicating the Arduino Board with the LCD.
4. If all else fails, reload the code developed by last year's code and start from a working standpoint.

6.1.5.2 Technical Risk 2: LEDs do not turn on

Description

The LEDs need to indicate whether data is being acquired or saved. If they do not turn on, this information cannot be seen by the user.

Probability: < Very Low, Low, Moderate, High, or Very High>

Low: The LEDs have already been tested with code and prove to turn on and off. Also, the voltage being supplied by the Arduino board can never surpass the maximum voltage the LED can handle; therefore, they cannot be burnt out.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Moderate: If the LEDs do not turn on, the user will not be able to visually see whether the data is being saved or acquired. However, this does not actually affect the process within the Arduino Board.

Strategy

1. Verify that the code loaded onto the Arduino Board does not contain any errors by using debugging tools.

6.1.5.3 Technical Risk 3: Low brightness level of LCD

Description

The team will hopefully be able to implement a scrolling wheel that allows the user to manually adjust the brightness of the LCD screen. This means that the brightness will not be static at a certain level as it was last year.

Probability: < Very Low, Low, Moderate, High, or Very High>

Moderate: Many factors can play into the adjustments of the brightness level. The code, potentiometer, and wiring can all heavily affect this risk.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Minor: The brightness level of the LCD screen will not be a factor if the user intends to use the device in the direct sunlight. Even if being used in a football stadium contained in a dome, the lighting should be sufficient.

Strategy

1. Debug the code that corresponds to the brightness level adjustments.

2. Check the wiring and make sure that there are no loose connections with the potentiometer.
3. Remove the feature as a whole and set the brightness level to a static condition.

6.1.5.4 Technical Risk 4: Push Buttons are dysfunctional

Description

The push buttons should tell the Arduino board to either save data or clear the current measurement displayed on the LCD.

Probability: < Very Low, Low, Moderate, High, or Very High>

Low: The push buttons were already configured by last year's team and worked properly.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Severe: If the push buttons do not work, the user will lose control of the interface. The user will not be able to clear or save data which defeats the purpose of the entire Impact Tester.

Strategy

1. Debug the code corresponding to the push buttons.
2. Reload the code implemented by last year's team.

6.1.6 Technical Risk: Power System

6.1.6.1 Technical Risk 1: Battery does not work/malfunctions

Description

The battery, though delivered, has yet to be tested at the time of this milestone.

Probability: < Very Low, Low, Moderate, High, or Very High>

LOW: The battery has been bought from a reputable company with positive reviews with the Better Business Bureau so there should be no issues with a defective battery or shady business. The possibility of the battery to stop working after frequent use is also there.

Consequences: <Minor, Moderate, Severe or Catastrophic>

SEVERE: Without the battery, the Impact Tester would have no means for it to be powered. While other forms of power could be used, such as replaceable batteries, the power supply that has been designed is based on the specific battery that has been purchased. It would be less severe the earlier the malfunction took place.

Strategy

1. Check power supply to make sure nothing would affect the battery.
2. Make sure the battery is not mishandled in the course of the project.

3. Frequent testing should be done to make sure no malfunctions have taken place.

6.1.6.2 Technical Risk 2: Battery charger does not work/malfunctions

Description

The risk that the battery charger does not work or malfunctions is very real. The charger could fail to supply the battery with the power it needs thus rendering the battery useless once discharged.

Probability: < Very Low, Low, Moderate, High, or Very High>

LOW: The battery has been bought from a reputable company with positive reviews with the Better Business Bureau so there should be no issues with a defective charger or shady business. The possibility of the charger to stop working/malfunction after frequent use is also there.

Consequences: <Minor, Moderate, Severe or Catastrophic>

MINOR: A replacement charger could always be purchased. This would set the schedule back once the battery discharges depending when the problem is caught. IT would not keep the prototype from functioning but would delay some time.

Strategy

1. Check on the charge time of the battery to make sure the charger gives the battery a consistent rate of charge.
2. Make sure that the charger is being used as instructed and handled with care.

6.1.6.3 Technical Risk 3: Failure of power supply

A failure in the power supply is a considerable risk to the project. The term “power supply” is referring to the main circuit board that supplies each component with the necessary voltage and current for it to function.

Probability: < Very Low, Low, Moderate, High, or Very High>

MODERATE: Though much attention in being put into the power supply board, there is always the risk of an error in calculation or a malfunction of a part on the power supply board.

Consequences: <Minor, Moderate, Severe or Catastrophic>

CATASTROPHIC: The power supply is what provides each component the power it needs to run. It is the main part of the prototype, without a working power supply the prototype is useless. Once again, the earlier that any potential risks or errors are discovered, the less risk there is to the overall project.

Strategy

1. Thorough testing of the power supply board through use of schematics

6.7.1 Technical Risk: Printed Circuit Board

6.7.1.1 Technical Risk 1: Unnecessary Components

Description

A component is disposed of because it seemed unimportant, but later on seems to be a crucial part of the senior design

Probability: < Very Low, Low, Moderate, High, or Very High>

High: There are many components in a microprocessor, and the functionality of each component can sometimes be difficult to decipher.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Severe: This could be a serious problem if it happens later in the spring semester.

Strategy

1. Fully analyze the component so that they can accurately pinpoint all unnecessary components.
2. Seek professional help from advisors to complete the analysis.
3. Leave all questionable components just in case they are crucial.

6.7.1.2 Technical Risk 2: Manufacturing Time

Description

Manufacturing time is longer than for casted

Probability: < Very Low, Low, Moderate, High, or Very High>

Moderate: It depends on the amount of customers a company has and the amount of merchandise the company trades

Consequences: <Minor, Moderate, Severe or Catastrophic>

Moderate: The longer the board takes to manufacture, the longer the team is in a standstill to test the prototype board

Strategy

1. Get quotes from different manufacturers.
2. Have many tasks unrelated to the manufacturing for the team to do in the meantime.
3. Preorder the components early so no time is wasted if more than one manufacturer used.

6.7.1.3 Technical Risk 3: Improper Design

Description

The board was not designed properly.

Probability: < Very Low, Low, Moderate, High, or Very High>

Low: Designing guidelines can be extensive. Since there are so many different guidelines to remember it may become easy to forget certain aspects of the design.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Moderate: Consequences are considerably lessened if mistakes are caught early on. Prolonging the find can cause severe consequences, especially if the board is already designed

Strategy

1. Guidelines in the “ICC-2221 Generic Standard on Printed Board Design” will be used for accurate designs
2. Work on boards will be extensively checked for mistakes periodically
3. Design will be checked by multiple members in order to ensure no mistakes were made

6.7.1.4 Technical Risk 4: Manufacturing Error

Description

There was a mistake made during board manufacturing

Probability: < Very Low, Low, Moderate, High, or Very High>

Very Low: PCB manufacturing is a large business that has been going on for many years. Most companies have legacies and have nearly perfected all manufacturing aspects.

Consequences: <Minor, Moderate, Severe or Catastrophic>

Severe: If there was an error in manufacturing and debugging discovers it, this could drastically delay the project due to a loss of time during debugging.

Strategy

1. Send all files to the manufacturer needed to create a working board
2. Use footprints for components in order to ensure correct polarization
3. Speak to the manufacturer in case any question arise
4. Chose a prestigious company in this field of business

6.2 Schedule Risks

When creating a schedule, it is important to have time allotted to accomplish each task. When a problem arises, the project will be less likely to become delayed because the amount of time needed to fix any problem that arises has already been taken into account in the schedule. This is an effective way to create a schedule and make sure the team is where they intended to be at this point in time.

6.2.1 Schedule Risk: Behind Schedule

Description

The team is behind schedule or running out of time

Probability: < Very Low, Low, Moderate, High, or Very High>

HIGH: Due to the fact that so much is going on along with class time there is a chance that a multitude of things can go wrong which can cause the group to be behind schedule.

Consequences: <Minor, Moderate, Severe or Catastrophic>

SEVERE: The progress of the project will delay and ultimate delay other deadlines needed to be met.

Strategy

1. Order components once it is known which parts are going to be used
2. Find components that won't require additional coding
3. Find components similar to those in existing prototype
4. Find components that will ship promptly
5. **Work overtime**

6.2.2 Schedule Risk 2: Need to re-order a damaged component

Description

One of the components broke and need to be re-ordered.

Probability: < Very Low, Low, Moderate, High, or Very High>

MODERATE: Besides for the case of the accelerometer (then it would be HIGH), the majority of the components have been left over from last year and are in good working condition.

Consequences: <Minor, Moderate, Severe or Catastrophic>

SEVERE: The progress of the project will delayed and ultimate delay other deadlines needed to be met.

Strategy

1. Make sure that everything is tested on a bread board before so everything can be managed before permanently placed.
2. If a component damages, re-order it
3. Keep working on other parts of the project
4. **When the new component comes in work overtime**

6.2.3 Schedule Risk 3: Prototype isn't finished prior to winter break

Description

Phase one: upgrading the prototype isn't finished before our first deadline (winter break)

Probability: < Very Low, Low, Moderate, High, or Very High>

MODERATE: Programming issues can result in longer work hours and ultimately more work days.

Consequences: <Minor, Moderate, Severe or Catastrophic>

MODERATE: The progress of the project will delayed, however, if the prototype isn't finish before winter break there is always the break to work on it.

Strategy

1. Make sure to get everything done before
2. **Work during winter break**

6.2.4 Schedule Risk 4: The transition from one component to another will not be smooth.

Description

When looking into new components to replace already existing components, one runs the risk of these new components not functioning with the other components in the prototype. This can be a major risk that can highly delay the project because no further progress will be able to be met until the problem can be resolved. This type of problem can also be difficult to pinpoint because this is usually a software problem in where new code must be written or some previously existing code must be modified. This debugging process can take up a lengthy amount of time. Ways to avoid a situation like this would be to read the data sheets for the new components. Many components are Arduino friendly since Arduino is an open-source microcontroller used by customers around the world. This will hopefully make this risk less likely than others.

Probability: < Very Low, Low, Moderate, High, or Very High>

VERY HIGH: Even though the software may be similar between components (for example, the Arduino will still have the same language to call on the pins and the GPS uses NMEA protocol), there are large chances that since the components themselves are different that they will not function properly.

Consequences: <Minor, Moderate, Severe or Catastrophic>

SEVERE: The progress of the project will delayed and more coding will need to be implemented in order to continue further into testing.

Strategy

1. Research the proper programming language that will interact with the components
2. Use the code already written and debug what is necessary
3. If old code fails (including debugging it), write new code
4. **Work overtime**

6.3 Budget Risk

Budgetary risk may happen when one does not take into account problems that may arise such as faulty components. Although it is hard to foresee this problem happening, it can happen. It is important to know what steps to take when something like this happens because if not done correctly, the delay time can become substantial.

6.3.1 Budget Risk 1: More components must be purchased

<u>Description</u>

There is a chance that either certain components are missing and will have be purchased, yet there might be not funds to do so

<u>Probability: < Very Low, Low, Moderate, High, or Very High></u>

LOW: The majority of the components have either been purchased or will not exceed our current budget if purchased. Also, all mechanical work is done by Mr. Mascaro himself.

<u>Consequences: <Minor, Moderate, Severe or Catastrophic></u>

LOW: More funding will have to be requested or money will have to be raised

<u>Strategy</u>

4. Pricing strategies will be looked into to make sure the design team is picking components that are efficient and price effective
5. Locating more funds will be further looked into if needed

6.3.2 Budget Risk 2: The price for a component was underestimated

<u>Description</u>

The price for a component could cost more than previously expected.

<u>Probability: < Very Low, Low, Moderate, High, or Very High></u>

MODERATE: This is considered moderate for the majority of the components except the accelerometer it's HIGH. This is because all our components are those that prices are pretty set for, however, the accelerometer is an expensive component itself and the price can vary depending where it is purchased from.

<u>Consequences: <Minor, Moderate, Severe or Catastrophic></u>

MODERATE: Higher budget than previously accounted for.

<u>Strategy</u>

1. Create room in the budget for something like this to happen
2. Check for components that are similar to the one being looked at yet don't break the budget.

6.3.3 Budget Risk 3: Previous component needs to be re-ordered

<u>Description</u>

One of the components broke, is faulty, or does not function the way previously thought and needs to be re-ordered.

Probability: < Very Low, Low, Moderate, High, or Very High>

MODERATE: The majority of the components and their functions are known or have been ordered already. HIGH: The accelerometer has a high chance of blowing.

Consequences: <Minor, Moderate, Severe or Catastrophic>

MODERATE: The progress of the project will delayed and ultimate delay other deadlines needed to be met.

Strategy

1. Make sure the item is highly reliable so something like this won't happen again
2. Read the datasheet and check to see if the problem can be fixed
3. Check the return policy of the item
4. Purchase another component such that more time is not wasted waiting

7 Conclusion

The Turf-Tec International Senior Design team #4 will deliver a finalized Impact Tester product to the sponsor, Mr. Mascaro. The product will bring to a completion the work of the previous year's teams and produce a market ready Impact Tester. The biggest contribution added this year will be the new Printed Circuit Board. The new design will allow for a more organized design which should create an easier building method when the product is being mass produced. That design is completed and ready to be sent to be constructed. Upon receiving the new PCB the task of beginning testing will take place and has been planned for. The team is already accustomed to working with the Printed Circuit Boards and the Arduino IDE because of the materials left over from the previous year's project so once the new PCB arrives the team should be ready to begin work on implementing the design. The power system has already been designed and tested and all parts received the correct amount of power needed. The code has been successfully loaded onto the board so the task of completing the software will be done. Once the PCB arrives and the impact tester is put together testing the software will be more easily done. All other parts are accounted for and have been tested to make sure they are in working order.

All factors into completing the project have been accounted for and the team is more than confident that we will be able to deliver a working impact tester. The efforts of the previous year's teams have provided a great foundation for which to continue the project on. The efforts of Dr. Frank and Mr. Mascaro are also to be appreciated.

The final deliverable will be the working impact tester itself. However, the team will deliver a fully designed Printed Circuit Board that will incorporate all of the needed hardware for the board. The Turf-Tec team will document all of the work in a final report and will deliver a manual containing the instructions for operating the Impact Tester. The manual documentation will instruct the user in detail on every relevant aspect of the Impact Tester, from using the user interface on the device to what the readings on the LCD mean and more.

8 References

All references have been referenced within the text